PENGEMBANGAN *STATEFUL PACKET INSPECTION* DENGAN METODE NDPMON UNTUK *DUPLICATE ADDRESS DETECTION*

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Steven Urbani
NIM: 145150200111158

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2018

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

PENGESAHAN

PENGEMBANGAN STATEFUL PACKET INSPECTION DENGAN MEKANISME IDS UNTUK DUPLICATE ADDRESS DETECTION

SKRIPSI

Untuk memenuhi sebagian persyaratan

memperoleh gelar Sarjana Komputer

Disusun Oleh :

Steven Urbani

NIM: 145150200111158

Skripsi ini telah diuji dan dinyatakan lulus pada
9 november 2018

Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Widhi Yahya, S.Kom, M.Sc.	**Eko Sakti Pramukantoro, S.Kom, M.Kom.**
NIK. 201607 870423 1 002	**NIK. 201102 860805 1 001**

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D
NIP. 19710518 200312 1 001

IDENTITAS TIM PENGUJI

PENGEMBANGAN *STATEFUL PACKET INSPECTION* DENGAN MEKANISME IDS UNTUK *DUPLICATE ADDRESS DETECTION*

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Steven Urbani
NIM: 145150200111158

Skripsi ini telah diuji dan dinyatakan lulus pada
9 November 2018
Telah diperiksa dan disetujui oleh:

Dosen Penguji I	Dosen Penguji II
Reza Andria Siregar, S.T., M.Kom.	Mahendra Data, S.Kom., M.Kom.
NIP. 19790621 200604 1 003	NIK. 2015038611171001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata di dalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, Oktober 2018

Steven Urbani

NIM: 145150200111158

KATA PENGANTAR

Puji syukur peneliti panjatkan kepada Tuhan Yang Maha Esa atas limpahan rahmat dan petunjuk-Nya, sehingga peneliti dapat menyelesaikan skripsi yang berjudul "*PENGEMBANGAN STATEFUL PACKET INSPECTION DENGAN METODE NDPMON UNTUK DUPLICATE ADDRESS DETECTION*".

Dalam penyusunan dan penelitian skripsi ini tidak lepas dari bantuan moral dan materiil yang diberikan dari berbagai pihak, maka peneliti mengucapkan banyak terima kasih kepada:

1. Bapak Wayan Firdaus Mahmudy, S.Si, M.T, Ph.D. selaku Dekan Fakultas Ilmu Komputer Universitas Brawijaya Malang.
2. Bapak Heru Nurwarsito, Ir., M.Kom. selaku Wakil Ketua I Bidang Akademik Fakultas Ilmu Komputer Universitas Brawijaya Malang.
3. Bapak Tri Astoto Kurniawan, S.T, M.T, Ph.D. selaku Ketua Jurusan Teknik Informatika Universitas Brawijaya Malang.
4. Bapak Agus Wahyu Widodo, S.T, M.Cs. selaku Ketua Program Studi Teknik Informatika Universitas Brawijaya Malang.
5. Bapak Widhi Yahya, S.Kom, M.Sc. selaku dosen pembimbing I yang telah memberikan pengarahan dan bimbingan kepada peneliti, sehingga dapat menyelesaikan skripsi ini dengan baik.
6. Bapak Eko Sakti Pramukantoro, S.Kom, M.Kom. selaku dosen pembimbing II yang telah memberikan pengarahan dan bimbingan kepada peneliti, sehingga dapat menyelesaikan skripsi ini dengan baik.
7. Kedua orang tua dan seluruh keluarga besar atas segala nasehat, kasih sayang, perhatian, dan kesabarannya memberikan semangat kepada peneliti, serta senantiasa tiada hentinya memberikan doa demi terselesaikannya skripsi ini.
8. Seluruh civitas akademika Informatika Universitas Brawijaya dan terkhusus untuk teman-teman Teknik Informatika Angakatan 2014.

Peneliti menyadari bahwa dalam penyusunan skripsi ini masih terdapat banyak kekurangan, sehingga saran dan kritik yang membangun sangat peneliti harapkan. Akhir kata peneliti berharap skripsi ini dapat membawa manfaat bagi semua pihak yang membutuhkannya.

ABSTRAK

Software defined networking merupakan bidang penelitian baru yang sedang berkembang, dimana konsep tersebut bertujuan untuk menggantikan jaringan yang tidak fleksibel dan rumit pada saat ini dengan jaringan yang inovatif dan lincah, namun mengusung manejemen jaringan yang lebih efisien dengan memisahkan *control plane* dan *data plane*. Salah satu protokol yang dapat mengusung konsep SDN adalah OpenFlow. Permasalahan tentang keamanan jaringan terkait suatu serangan maupun celah kerap terjadi pada jaringan OpenFlow. Salah satu serangan pada jaringan OpenFlow adalah IPv6 *duplicate address detection* DoS, dimana penyerang memanfaatkan celah pada skema pengecekan duplikasi alamat IPv6 pada suatu jaringan lokal. Salah satu program yang dapat diterapkan pada kontroler untuk mengatasi permasalahan keamanan pada OpenFlow adalah *Stateful Packet Inspection*. Dalam mangatasi celah pada *duplicate address detection*, dilakukan pengembangan terhadap *Stateful Packet Inspection* agar dapat mencegah terjadi serangan pada proses *duplicate address detection* yang dilakukan oleh penyerang dengan menambahkan mekanisme pengecekan NDP dengan metode NDPMon. Terdapat beberapa pengujian terhadap sistem untuk menguji fungsionalitas serta kinerja dalam mengatasi celah pada proses *duplicate address detection* pada jaringan OpenFlow. Dari hasil pengujian yang dilakukan, sistem mampu mengenali proses *DAD* serta berhasil manangani serangan dari tiga skenario pengujian *DAD DoS*.

repository.ub.ac.id

ABSTRACT

Software defined networking is a new field of research, where the concept aims to replace an inflexible and complex network with an innovative and agile network by separating the control plane and data plane. One of the protocols that can carry the concept of SDN is OpenFlow. Problems about network security that related about gap on conventional network are also occur on the OpenFlow network. One of the network attacks which can be threatening the security of the OpenFlow network is DoS of duplicate address detection, which attackers exploit a gap on IPv6 duplication checking schemes in a local network. Program that can be applied on controller to resolve some security issues in OpenFlow is Stateful Packet Inspection. To resolving the gap, development of the Stateful Packet Inspection can be done to prevent attacks on duplicate address detection processes by adding an NDP checking mechanism using the NDPMon method. There are several tests to test the system's functionality and performance in overcoming gaps in DAD process on the OpenFlow network. From the test results, we obtained that the system can noticed duplicated address detection process and can handled attacker from three test scenarios for DAD DoS.

UNIVERSITAS BRAWIJAYA

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Software defined networking merupakan bidang penelitian baru yang sedang berkembang, dimana konsep tersebut bertujuan untuk menggantikan jaringan yang tidak fleksibel dan rumit pada saat ini dengan jaringan yang inovatif dan lincah, namun mengusung manejemen jaringan yang lebih efisien dengan memisahkan *control plane* dan *data plane* (Vipin Gupta, 2016). Salah satu protokol yang dapat digunakan dalam mewujudkan konsep SDN adalah protokol OpenFlow, dimana OpenFlow dapat mengatur komunikasi antara *control plane* dan *data plane* secara terpisah. Perkembangan SDN dan OpenFlow tidak sepenuhnya berjalan baik, dimana konsep SDN tidak sepenuhnya menyelesaikan permasalahan yang ada pada jaringan tradisional. Aspek utama yang menjadi penghambat perkembangan SDN adalah kerentanan pada keamanan jaringan, dimana konsep SDN tidak mengatasi celah keamanan yang telah ada pada jaringan tradisional (Wajdy Othman, 2017).

Salah satu serangan yang dapat dilakukan pada jaringan OpenFlow adalah dengan mengeksploitasi celah pada NDP (*Neighbour Discovery Protocol*) IPv6. NDP merupakan protokol yang dijalankan *node* berbasis IPv6 dalam proses pencarian *node* lain, identifikasi alamat *node* lain serta mengetahui *router* dalam satu lapisan *link* (H. Singh, 2010). Salah satu celah yang terdapat pada proses yang memanfaatkan NDP IPv6 adalah *duplicate address detection denial of service*. *DAD (Duplicate address detection) DoS* merupakan suatu serangan yang mengeksploitasi proses *DAD* pada IPv6 dengan memanfaatkan NDP. *DAD* merupakan proses pengecekan alamat IPv6 pada *node* baru yang ingin terhubung ke jaringan lokal IPv6 dengan memanfaatkan NDP yang bertujuan untuk menghindari terjadinya duplikasi pada alamat lokal IPv6 yang digunakan dalam satu jaringan. Pada serangan *DAD DoS*, penyerang mengirimkan *neighbor advertisement* yang telah dimodifikasi untuk merespon paket *broadcast neighbor advertisement* yang dikirim oleh *node* baru, dengan tujuan tiap *node* baru tidak dapat terhubung (tidak mendapatkan alamat IPv6) ke jaringan lokal IPv6, sehingga *node* baru tersebut mengalami *denial of service*. (Shafiq Ul Rehman, 2016).

Terdapat beberapa penelitian yang dilakukan dalam penyelesaian masalah. Tomas Hegr, dalam penelitiannya menerapkan OpenFlow Security Engine (OFSE) pada Floodlight *controller*, dimana OFSE dapat memantau paket IPv6 dari koneksi yang aktif untuk mencegah terjadinya *DoS* melalui serangan *duplicated address detection* IPv6 (Tomas Hegr, 2013). Terdapat penelitian lainnya dalam mengatasi serangan IPv6 *DoS* pada jaringan OpenFlow, dimana dalam mencegah terjadinya serangan, peneliti menerapkan *traffic statistics gathered by* OpenFlow *port policies*

control, sehingga sistem mampu mencegah IPv6 *DoS* dengan menerapkan *whitelist policies* pada koneksi yang aktif berdasarkan parameter yang telah ditentukan (Chia-Wei Tseng, 2014). Penelitian pendukung lainnya yang mengangkat permasalahan keamanan pada jaringan OpenFlow menerapakan *stateful* OpenFlow *firewall* pada *POX controller* yang mampu menyaring paket IPv4 dan menyimpan informasi paket IPv4 pada tabel, dimana penelitian tersebut berhasil mengatasi *spoofed packet* yang menggunakan protokol IPv4 (Vipin Gupta, 2016). Namun pada penelitiannya, sistem belum mampu menangani serta menyaring paket dengan protokol IPv6, dimana struktur serta protokol translasi *link-layer* dalam penentuan jalur berbeda bila dibandingkan dengan protokol IPv4 (Aftab Siddiqui, 2017). Sistem yang dikembangkan pada penelitian tersebut belum mampu mengatasi celah pada NDP IPv6, khususnya pada proses *DAD*, dimana sistem tersebut hanya terbatas pada inspeksi paket dengan protokol paket TCP.

Dari permasalahan serta batasan pada penelitian sebelumnya, dilakukan penelitian terhadap pengembangan *Stateful Packet Inspection* yang mampu melakukan penyaringan paket berbasis protokol IPv6, khususnya pada proses *DAD*. Pengembangan sistem dilakukan dengan menerapkan mekanisme penanganan paket yang disesuaikan berdasarkan tipe paket ICMPv6 yang masuk. Mekanisme yang digunakan mengadaptasi cara kerja IDS (Intrusion Detection System) yang dikembangkan oleh Ferdous Barbhuiya, dimana sistem memiliki mekanisme NDPMon (*Neighbor* Discovery Protocol-Monitor) untuk melihara informasi dari tiap paket NDP unik yang masuk berdasarkan waktu diterimanya paket (Ferdous Barbhuiya, 2013). Informasi yang diperoleh dari hasil inspeksi NDP akan dijadikan parameter dalam pembentukan *state table* dan ACL yang akan disimpan oleh kontroler. Hasil dari inspeksi NDP akan digunakan dalam pembentukan *flow rules* yang disimpan pada *switch* OpenFlow. Pengujian terhadap pengembangan sistem dilakukan dengan menjalankan skenario serangan *DAD* DoS untuk mengetahui kinerja dari sistem. Dengan demikian, sistem mampu mencegah terjadinya serangan *DAD DoS* pada jaringan OpenFlow, sekaligus mampu mengefisienkan proses pembentukan *flow rules* berdasarkan hasil dari penyaringan paket IPv6.

1.2 Rumusan Masalah

Berdasarkan penjelasan dari latar belakang diatas, dapat dirumuskan permasalahan sebagai berikut.

1. Bagaimana pengembangan *Stateful Packet Inspection* agar mampu mengatasi IPv6 *DAD Denial of Service* pada jaringan OpenFlow?
2. Bagaimana hasil uji pengembangan *Stateful Packet Inspection* terhadap serangan IPv6 *DAD Denial of Service* pada jaringan OpenFlow?

1.3 Tujuan

Berdasarkan penjabaran rumusan masalah, dapat ditentukan tujuan penelitian sebagai berikut.

1. Mengetahui bagaimana pengembangan *stateful packet* agar mampu mengatasi IPv6 *DAD Denial of Service* pada jaringan OpenFlow?
2. Mengetahui hasil uji pengembangan *Stateful Packet Inspection* terhadap serangan IPv6 *DAD Denial of Service* pada jaringan OpenFlow?

1.4 Manfaat

Adapun manfaat penelitian yang diharapkan adalah sebagai berikut.

- **Manfaat bagi penulis**

Dapat meningkatkan dan memantapkan pengetahuan teoritis maupun aplikatif terhadap kemanan pada jaringan OpenFlow.

- **Manfaat bagi peneliti lain**

Dapat menjadi acuan ilmiah untuk penelitian tentang *Stateful Packet Inspection* berbasis protokol IPv6 pada jaringan OpenFlow.

- **Manfaat bagi jaringan**

Dapat mengoptimalkan utilitas dari suatu jaringan serta meningkatkan keamanan pengiriman data pada jaringan OpenFlow.

1.5 Batasan Masalah

Agar pembahasan masalah dapat terarah dengan baik dan tidak menyimpang dari pokok permasalahan, maka penulis membatasi permasalahan yang akan dibahas, yakni:

1. Pengembangan sistem serta serangan yang diuji terhadap sistem hanya dilakukan IPv6 *DAD denial of Service*.
2. Dilakukan dalam lingkungan emulasi jaringan Mininet dengan perulangan pengujian sebanyak 10 kali pada tiap skenario serangan.
3. Setiap *host* menggunakan konfigurasi IPv6 *State*less Address Autoconfiguration (SLAAC) atau alamat tiap *host* telah dikonfigurasi sebagai IPv6 statis.
4. Penelitian dilakukan dengan menggunakan 5 host dan 2 switch, sehingga informasi serta analisis performa sistem terhadap jumlah node yang masif belum didapatkan.

1.6 Sistematika Laporan

Uraian singkat mengenai struktur penulisan pada masing-masing bab adalah sebagai berikut.

BAB 1 PENDAHULUAN

Bab ini menggambarkan latar belakang, rumusan masalah, tujuan berdsarkan rumusan masalah, manfaat dari penelitian, penentuan batasan penelitian serta sistematika penulisan dari penelitian.

BAB 2 LANDASAN KEPUSTAKAAN

Bab ini mencakup kajian dari penelitan sebelumnya yang memiliki keterhubungan dan menampilkan berbagai teori serta konsep yang diperoleh dari sumber penelitian yang memilki keterikatan yang nantinya digunakan sebagai panduan dalam penelitian.

BAB 3 METODOLOGI

Bab ini memaparkan metodologi yang dipakai dalam penelitian. Bab ini berisi penjelasan dari langkah penelitian yang terdiri dari studi pustaka, analisis kebutuhan, perancangan sistem, implementasi, pengujian serta pengambilan kesimpulan.

BAB 4 PERANCANGAN

Bab ini berisi tentang tahap-tahap perancangan yang dilakukan untuk membangun IPv6 *Stateful Packet Inspection* pada pada jaringan OpenFlow berdasarkan metodologi dan analisis kebutuhan yang telah dilakukan.

BAB 5 IMPLEMENTASI

Bab ini berisi implementasi dari IPv6 *Stateful Packet Inspection* pada jaringan OpenFlow yang telah dirancangkan sebelumnya.

BAB 6 PENGUJIAN

Bab ini menampilkan hasil dari pengujian sistem IPv6 *Stateful Packet Inspection* berdasarkan batasan pengujian serta skema pengujian yang telah di rancang.

BAB 7 PENUTUP

Bab ini memaparkan penarikan kesimpulan dari hasil penelitian dan perancangan sistem, serta pemberian saran untuk pengembangan di kemudian hari, agar dapat dilakukan pengembangan di masa depan.

repository.ub.ac.id

UNIVERSITAS BRAWIJAYA

BAB 2
LANDASAN KEPUSTAKAAN

2.1 Kajian Pustaka

Pada subbab ini dilakukan kajian terhadap penelitian yang telah ada sebelumnya dan terkait dengan penelitian ini. Penelitian tersebut nantinya dijadikan sebagai pedoman dalam pelaksanaan penelitian sekarang. Berikut tabel perbandingan penelitian terdahulu dan sekarang:

Tabel 2.1 Kajian Pustaka

No	Nama Penulis, Tahun dan Judul	Persamaan	Perbedaan	
			Penelitian Terdahulu	Rencana Penelitian
1	Hegr, Tomas; Bohac, Leos; Uhlir, Vojtech; Chlumsky, Petr. (2013). OpenFlow *Deployment and Concept Analysis*	Mengatasi IPv6 *Duplicate address detection DoS* pada jaringan OpenFlow	Mengimplementasi OpenFlow *Security Engine* (OFSE)	Mengimplementasi *Stateful Packet Inspection* IPv6 pada OpenFlow
2	Tseng, Chia-Wei; Chen Sheue-Ji; Yang, Yao-Tsung. (2014). IPv6 *Operations and Deployment Scenarios over* SDN.	Mengimplementasi solusi pada SDN *controller* dalam mengatasi *DOS* IPv6	menerapkan *traffic statistics gathered by* OpenFlow *port policies control* untuk mengatasi *DoS*	Melakukan filtering IPv6 pada proses kerja protokol NDP (*Neighbor* Discovery Protocol) dalam mengatasi *DAD DoS*
3	Jayawardhana , Madusha; Perera, Ninada; Bandara, Priyankara. [2016]. *Implementation of Stateful* SDN *Firewall*	Mengimplementasi IPv6 *Filtering* pada SDN	Implementasi *stateful firewall untuk mengatasi* IPV4 *packet spoofing*	implementasi *Stateful Packet Inspection* dengan Ryu *controller* untuk mencegah *DAD DoS*

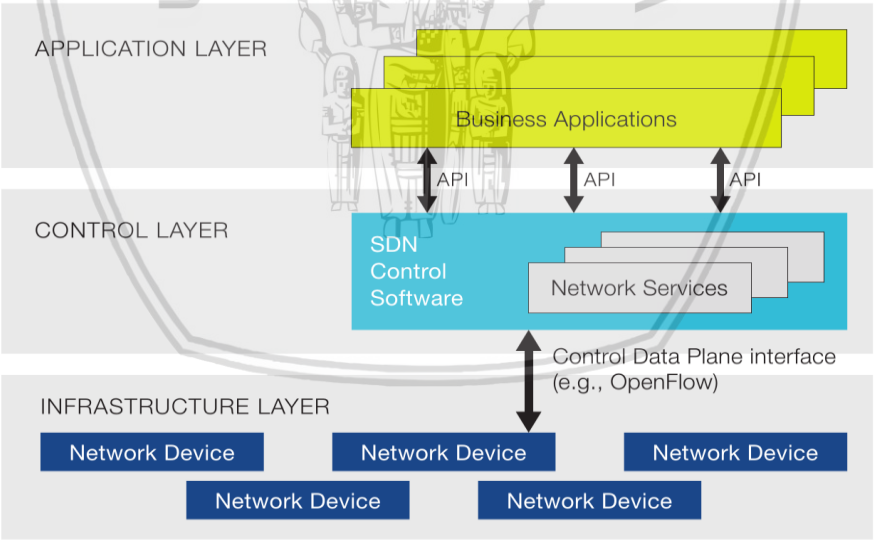
2.2 Dasar Teori

Pada subbab ini, dijabarkan berbagai teori yang dapat mendukung penelitian ini, yaitu berbagai teori untuk mengembangkan IPv6 *Stateful Packet Inspection* pada jaringan OpenFlow.

2.2.1 *Software-Defined Networking*

SDN merupakan sauatu konsep jaringan yang berbasis protokol OpenFlow. SDN memiliki arsitektur dimana jaringan membedakan antara *control plane*, yaitu komponen dari jaringan yang mengatur proses penerusan paket pada suatu jaringan, serta *data plane*, yaitu bagian yang menjalankan proses penerusan atau pengiriman paket dengan mengacu pada informasi yang diberikan oleh *control plane* (Kreutz, et al., 2015). Konsep dari SDN membuat jaringan lebih dinamis, mempermudah pengelolaan serta mampu beradaptasi lebih baik. Hal tersebut memungkinkan karena SDN memiliki *controller* yang dapat dimanipulasi secara langsung dan infrastrukturnya dapat diperbaharui dengan berbagai macam aplikasi serta layanan jaringan.

Melalui konsep *Software defined networking,* suatu jaringan yang sebelumnya bersifat terdistribusi tadi dapat dimanipulasi secara terpusat yang membuat proses penerusan paket dan pengaturan dari suatu jaringan menjadi lebih efisien (Astuto, Mendon¸ca, Nguyen, Obraczka, & Turletti, 2014).



Gambar 2.1 Arsitektur *Software-Defined Networking*

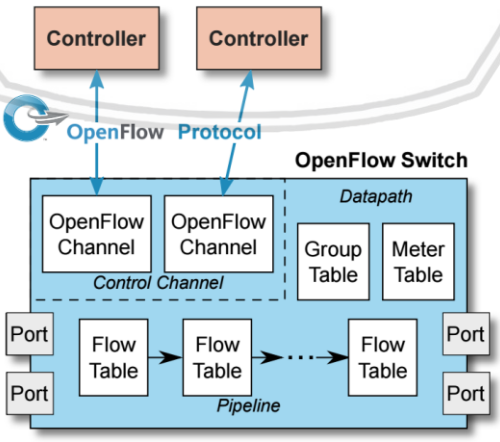
Sumber: sdxcentral.com (2015)

Menurut (Kreutz, et al., 2015), *Software defined networking* merupakan arsitektur jaringan yang memiliki 4 pilar, yaitu:

1. Pemisahan antara *control plane* dengan *data plane.* Kemampuan sebagai *control plane* disisihkan dari perangkat jaringan menjadi perangkat sederhana yang menerima serta meneruskan paket *(switch)*.
2. Aturan dalam meneruskan paket dilaksanakan bukan berdasarkan *destination* dari paket tersebut, melainkan berdasarkan *flow*. *Flow* dapat didefinisikan sebagai suatu kriteria/penyaring paket pada jaringan yang berdasarkan kapabilitas dari penerapanya (McKeown, et al., 2008).
3. Logika kontrol dipusatkan pada suatu *node* terpisah yang disebut *Controller* atau *Network Operating System*. NOS (*Network Operating System)* merupakan suatu platform perangkat lunak yang menyediakan *library, resource* serta abstraksi guna mendukung pemrograman logika kontrol yang dapat mengatur perilaku perangkat *data plane* (*switch*).
4. Jaringan dapat dimanipulasi dengan *software* yang berjalan diatas NOS serta dapat berkomunikasi dengan perangkat *data plane* yang dilingkupinya.

2.2.2 OpenFlow

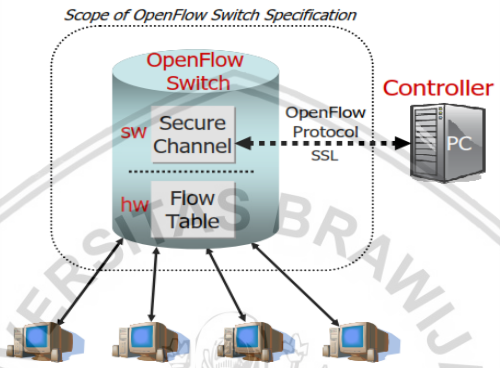
OpenFlow merupakan protokol komunikasi data pada jaringan yang dibangun diantara sisi kontroler (*control plane*) dan *switch* (*data plane*) pada arsitektur *software defined netwroking* (Open *Network*ing Foundation, 2016). OpenFlow mengemban *open-protocol* untuk memprogram *flow table* dari perangkat *data plane* yang beragam. OpenFlow menyediakan cara yang standar dan terbuka untuk sebuah *controller* SDN untuk berkomunikasi dengan sebuah *switch* yang terpasang OpenFlow. OpenFlow sendiri merupakan protokol jaringan bersifat terbuka dengan memakai *port* 6633 sebagai *default port* (McKeown, et al., 2008).



Gambar 2.2 OpenFlow pada Jaringan

Sumber: opennetworking.org (2015)

Cara kerja OpenFlow bersifat sentralis, dimana saat sebuah OpenFlow *switch* menerima paket baru yang unik dari paket sebelumnya dan tidak memiliki entri *flow* yang sama, maka *switch* tersebut akan mengirimkan informasi dari paket tersebut ke *controller*, dimana *controller* kemudian membuat keputusan untuk menangani paket tersebut. *Controller* dapat melakukan *action drop* pada paket tersebut serta mampu menambahkan entri *flow* yang dapat mengatur *switch/router* dalam proses penerusan paket yang serupa. (Open Networking Foundation, 2016).



Gambar 2.3 Arsitektur OpenFlow

Sumber: (McKeown, et al., 2008)

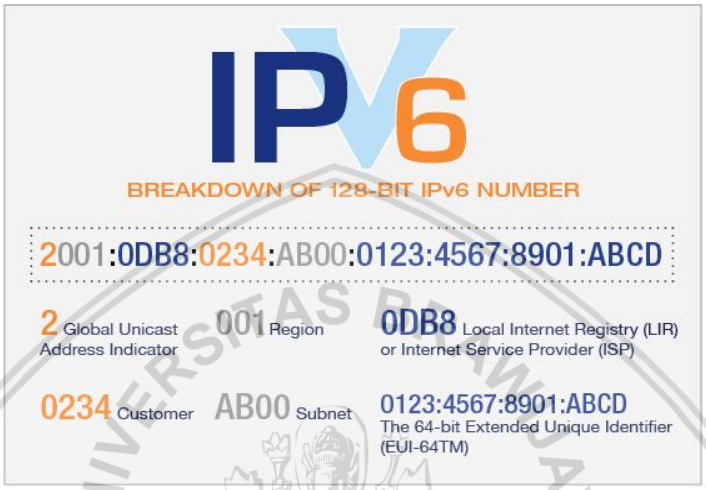
Pada *switch* OpenFlow, terdapat penghapusan akan fungsionalitas *control plane* sehingga *switch* OpenFlow diberikan tabel *flow* dari *controller* yang berisi set instruksi yang berlaku pada tiap paket. Instruksi yang ada pada *Switch* OpenFlow pada umumnya antara lain:

1. *Forwarding,* meneruskan paket dari suatu *port* menuju ke *port* tujuan paket.
2. Meneruskan paket ke *Controller.* Yang dijalankan pada saat paket pertama dari *flow* baru masuk agar *Controller* dapat mnentukan apakah *flow* tersebut dapat ditambahkan ke dalam *flow table* atau tidak.
3. *Drop*, membuang paket dari *flow* yang digunakan untuk meningkatkan keamanan jaringan sehingga paket yang mencurigakan tidak masuk ke dalam jaringan.

2.2.3 IPv6

IPv6 merupakan singkatan dari "*Internet Protocol Version* 6" yang merupakan protokol *Internet* generasi kedepan dan dirancang untuk menggantikan protokol *Internet* yang umumnya dipakai saat ini, yaitu IPv4. Agar dapat berkomunikasi melalui *Internet*, perangkat komputer diharuskan mempunyaii alamat asal dan alamat tujuan. Alamat numerik tersebut disebut sebagai alamat protokol *Internet*. Dikarenakan

perkembangan *internet* serta jumlah pengguna layanan *internet* bertambah secara cepat, maka kebutuhan terhadap alamat IP pun bertambah. Dengan protokol IPv6, maka ketersediaan akan alamat IP dapat diatasi dikarenakan IPv6 memiliki lebih banyak range IP yang lebih besar debandingkan IPv4 yang semakin haris semakin terbatas (*Cisco ACE Application Control Engine*, 2008).



Gambar 2.4 Arsitektur IPv6

Sumber: telehouse.com (2015)

Pada IPv4, panjang *bit* untuk satu alamat IP sebesar 32-*bit* yang dapat menyediakan 4,3 miliar alamat berbeda. IPv6 sendiri memilki memiliki panjang sebesar128-bit yang dapat dihitung jumlah alamat yang mungkin tersedia 2128=3,4 x 1038 alamat yang bertujuan untuk memberikan alamat yang tidak mudah habis kedepannya dengan tujuan mengurangi kompleksitas pada proses perutean secara luas (*Cisco ACE Application Control Engine*, 2008). IPv6 diproyeksikan akan menggantikan IPv4 yang semakin hari umlahnya semakin menipis. Namun untuk melakukan migrasi dari IPv4 ke IPv6 tidaklah mudah, dikarenakan sistem yang menggunakan IPv4 yang sudah ada sebelumnya tidak seluruhnya mendukung protokol IPv6.

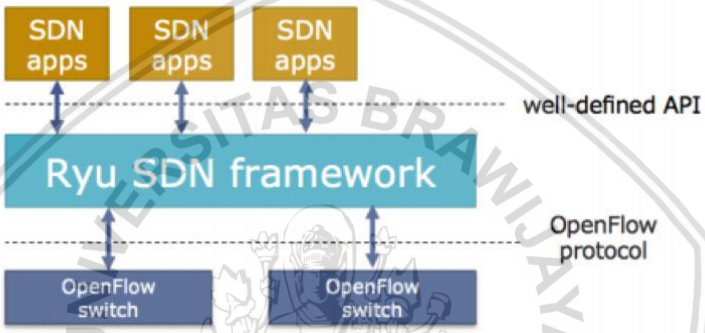
2.2.4 Mininet

Mininet merupakan salah satu *emulator* jaringan yang mendukung visualisasi jaringan yang memiliki beberapa komponen seperti *host*, *switch*, *controller* serta *link* (Mininet Team, 2016). Mininet mendukung simulasi untuk melakukan *prototyping* jaringan yang kompleks dengan memanfaatkan sumber daya yang terbatas seperti PC (Lantz, Heller, & McKeown, 2010). Mininet menggunakan virtualisasi berbasis proses pada suatu sistem operasi dalam membangun dan menjalankan banyak *host* dan *switch* pada topologi yang terbentuk. Mininet memanfaatkan *virtual software switch OpenvSwitch* dalam mendukung pembuatan topologi berbasis protokol OpenFlow.

Mininet dapat dijalankan dalam *mode Command Line Interface* maupun dalam *mode Graphical User Interface* yang biasa dikenal sebagai Miniedit.

2.2.5 Ryu

Ryu merupakan suatu *framework* berbasis Python untuk jaringan *software defined networking*. Ryu memberikan perangkat lunak serta API yang dapat membantu *developer* untuk membuat aplikasi kontrol jaringan dan manajemen jaringan (Ryu SDN *Framework* Community, 2014). Ryu dapat dibangun dengan memakasi bahasa pemrograman Python maupun dengan cara pengiriman pesan JSON dari *API* yang disediakan. Ryu mendukung beragam protokol untuk mengontrol jaringan, diantaranya: OpenFlow, Of-config, NetConf dan sebagainya.



Gambar 2.5 Kontroler Ryu

Sumber: sdxcentral.com (2014)

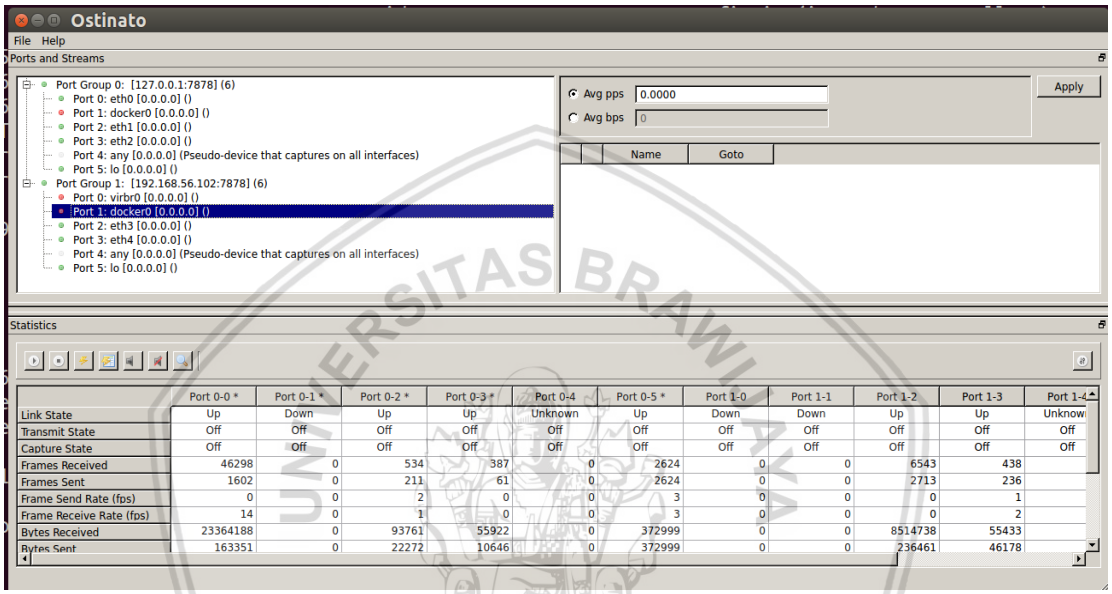
Kelebihan Ryu sebagai *framework* OpenFlow dikarenakan Ryu telah mendukung penggunaan OpenFlow versi 1.0 hingga OpenFlow 1.5. Ryu digunakan sebagai kontroler SDN dan sebagai *framework* yang nantinya dipadukan dengan pengembangan dari *Stateful Packet Inspection* pada protokol IPv6.

2.2.6 THC-IPv6 Toolkit

THC-IPv6 Toolkit merupakan kumpulan dari perangkat penguji jaringan berbasis protokol IPv6. Alat pengujian ini mendukung manipulasi paket berdasarkan alamat *ethernet*. IPv6 Toolkit memiliki dua *mode* operasi yaitu aktif, dimana alat ini mampu menjalankan sebuah serangan yang ditujukan kepada target yang spesifik seperti *spoofing, flooding*, *scanning*, serta *listening*, dimana alat ini mampu melakukan *listen* pada jaringan lokal dan memberikan paket serangan sebagai respon dari suatu trafik (Fernando Gont, 2012). *Tool* ini mampu memanipulasi paket IPv6 seperti jumlah alamat serangan, jumlah *port* yang digunakan, menentukan *port* dan alamat yang diserang serta modifikasi *frame rate* yang digunakan dalam serangan. pada sistem ini, nantinya *tool* ini digunakan dalam menerapkan penyerang IPv6 *DAD DoS* pada jaringan OpenFlow.

2.2.7 Ostinato *Network Traffic Generator*

Ostinato adalah generator dan penganalisis paket pada trafik jaringan dengan fitur *GUI* yang ramah serta dilengkapi *API* Python yang kuat untuk otomasi pengujian jaringan. Ostinato mampu membuat dan mengiirim paket pada beberapa aliran jaringan dengan protokol yang berbeda (Ostinato Network traffic Generator, 2018). Ostinato digunakan sebagai generator trafik jaringan dan alat tester jaringan yang dapat dimodifikasi sesuai kebutuhan dan keinginan pengguna.

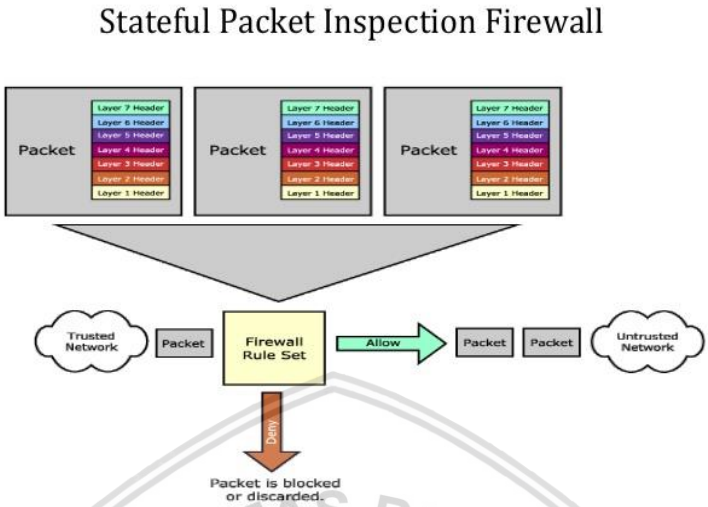


Gambar 2.6 Ostinato

Dengan Ostinato, pengembang jaringan dapat melakukan debug terhadap sistem jaringan dengan melakukan pengiriman paket modifikasi sesuai keadaan yang diinginkan. Ostinato sendiri mendukung penggunaan beberapa protokol familiar, seperti IPv4, IPv6, ICMP, dan lain sebagainya.

2.2.8 *Stateful Packet Inspection*

SPI merupakan sistem inspeksi paket yang tidak hanya melakukan pengecekan struktur paket serta data pada paket, SPI juga melakukan pengecekan keadaan pada tiap *host* yang saling berkomunikasi. SPI tidak hanya mengizinkan *firewall* untuk melakukan pengecekan isi pada paket yang masuk, SPI juga melakukan penampisan berdasarkan keadaan koneksi (*connection state*). Hal ini membuat sistem keamanan mempunyai keamanan yang lebih fleksibel serta mempunyai skalabilitas dalam hal penapisan yang lebih kompleks (Zhibin Zhang, 2007).



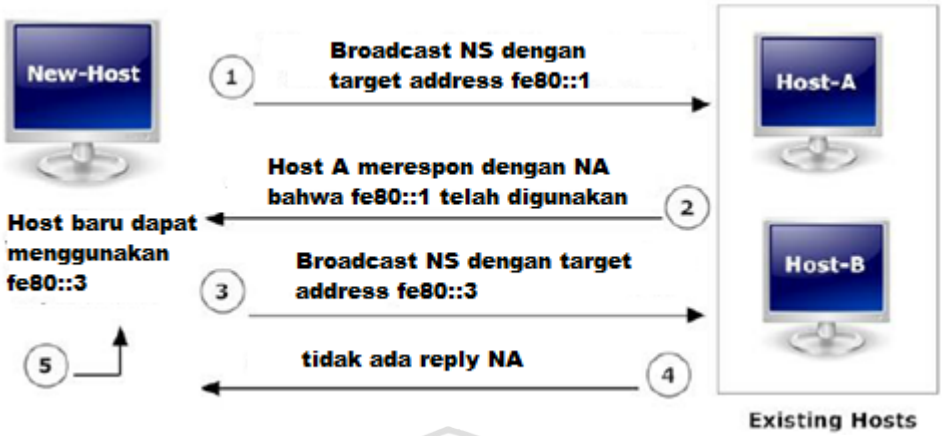
Gambar 2.7 Arsitektur *Stateful Packet Inspection*

Sumber: slideshare.net (2014)

Perbedaan utama dari *SPI* dengan paket inspeksi pada umumnya ialah saat suatu koneksi sudah dikenali serta diizinkan, pada dasarnya kebijakan (*firewall policy*) tidak lagi dibutuhkan untuk mengizinkan komunikasi balasan dikarenakan *firewall* telah mengetahui respon yang seharusnya diterima kemudian. *Firewall* dibangun hingga mampu membedakan paket yang sah sesuai kebijakan *firewall*. Hanya paket yang sesuai koneksi yang telah terbentuk dan dikenal saja yang akan diizinkan oleh sistem untuk melewatinya kearah jaringan privat.

2.2.9 *Dupicate Address Detection*

Duplicate address detection merupakan sebuah mekanisme yang memastikan bahwa tiap *host* yang terhubung pada suatu jaringan lokal memiliki alamat IPv6 yang unik dengan melakukan verifikasi antar *host*. Saat *Host* baru ingin terhubung ke jaringan lokal, *host* tersebut akan menanyakan apakah alamat IPv6 yang ingin dipakai oleh *host* tersebut tidak duplikasi alamat IPv6 dengan *host* lainnya (Shafiq Ul Rehman, 2016). Proses *DAD* ini memanfaatkan *NDP* (*Neighbor* Discovery Protocol), dimana *host* baru akan mengirim *broadcast* paket *neighbor solicitation* keseluruh *host* pada jaringan lokal untuk memberitahu alamat IPv6 yang ingin digunakan. Bila alamat IPv6 telah digunakan oleh *host* yang telah terlebih dahulu terhubung di jaringan, maka *host* yang menggunakan alamat tersebut akan mengirimkan paket *neighbor advertisement* kepada *host* baru untuk memberitahu bahwa alamat IPv6 tersebut telah digunakan.



Gambar 2.8 Proses *Duplicate address detection*

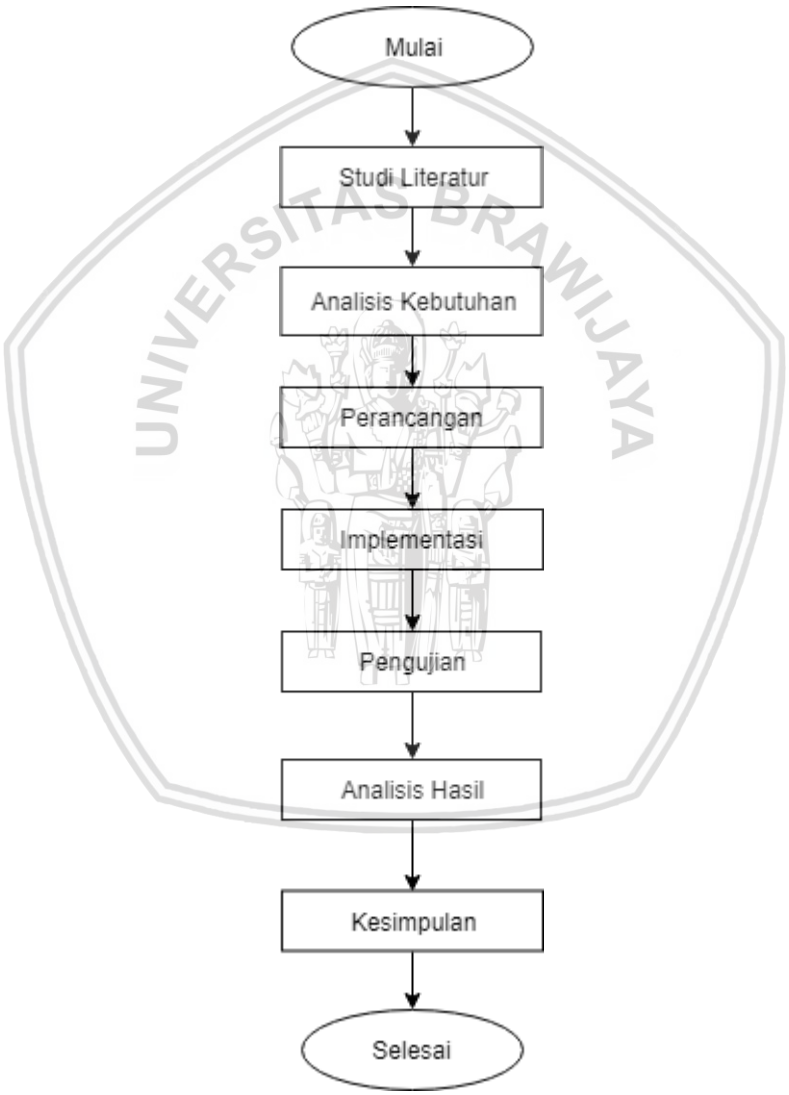
Gambar 2.8 menjelaskan proses *duplicate address detection* dimana *host* baru ingin terhubung ke jaringan yang sudah ada. Pada tahap pertama, *host* baru melakukan *broadcast neighbor solicitation* dengan menanyakan bahwa *host* tersebut ingin menggunakan "fe80::1". *Host* A merespon *broadcast* tersebut dengan mengirim *neighbor advertisement* kepada *host* baru bahwa IPv6 tersebut telah digunakan. Pada tahap 3, *host* baru kembali melakukan *broadcast neighbor solicitation* kembali dengan membawa informasi alamat IPv6 lainnya, yaitu "fe80::3", namun tidak ada balasan *neighbor advertisement* dari *broadcast neighbor solicitation* tersebut, sehingga *host* baru dapat menggunakan alamat IPv6 tersebut dan menandakan proses *duplicated address detection* selesai.

2.2.10 *Neighbor Discovery Protocol Monitoring (NDPMon)*

Neighbor Discovery Protocol Monitor (NDPMon) adalah mekanisme diagnostik yang digunakan oleh administrator jaringan IPv6 untuk memantau paket yang menggunakan protokol ICMPv6. NDPMon mengamati anomali yang terjadi pada jaringan lokal terhadap paket yang memanfaatkan Neighbor Discovery Protocol (NDP), terutama selama proses *stateless address autoconfiguration*. NDPMon juga memperbarui daftar *node* tetangga pada jaringan dan memantau perubahan informasi dari node yang memanfaatkan protokol ICMPv6. Hal tersebut memungkinkan administrator untuk melacak penggunaan IP-MAC serta informasi dari tiap *node IPv6*.

BAB 3
METODOLOGI

Pada bagian ini dijelaskan tentang metode dan langkah-langkah yang dilakukan dalam pengerjaan penelitian ini. Berikut merupakan tahapan-tahapan metodologi penelitian yang diambil, tergambar pada diagram alir yang ditunjukkan gambar 3.1.



Gambar 3.1 Diagram Alir Metode Penelitian

Berdasarkan gambar 3.1, diuraikan langkah-langkah metodologi penelitian yang akan dilakukan, yaitu:

1. Studi literatur penelitian sebelumnya yang terkait, *software-defined networking*, *Stateful Packet Inspection,* IPv6, NDP, Mininet, Ryu*,* dan *duplicate address detection.*
2. Analisis kebutuhan sistem yang meliputi kebutuhan fungsional dan non-fungsional.
3. Perancangan dasar, topologi, dan lingkungan dalam menerapkan *Stateful Packet Inspection* untuk protokol IPv6 pada jaringan OpenFlow.
4. Implementasi pengembangan *Stateful Packet Inspection* untuk menangani serangan IPv6 *DAD DoS*.
5. Pengujian pengembangan sistem serta analisis hasil pengujian berdasarkan parameter uji yang telah ditentukan.
6. Penarikan kesimpulan berdasarkan hasil analisis pengujian yang dilakukan terhadap sistem.

3.1 Studi Literatur

Studi literatur pada penelitian ini digunakan sebagai dasar serta landasan mengenai perancangan dan implementasi sistem. Beragam studi literatur yang dilaksanakan terhadap ilmu dasar yang meliputi IPv6*, Stateful Packet Inspection, duplicate address detection,* serta OpenFlow berikut aplikasi terkait mengenai penerapannya. Studi literatur terhadap dasar-dasar Ryu *controller* juga dibutuhkan guna pengembangan serta penerapan *Stateful Packet Inspection* IPv6 pada jaringan OpenFlow. Studi literatur terakhir dipaparkan untuk mengetahui berbagai jenis skema pengujian sistem yang cocok untuk digunakan pada penelitian ini.

3.2 Analisis Kebutuhan

Analisis kebutuhan dilakukan guna mencari tahu serta menganalisis aspek yang diperlukan dalam menjalankan penelitian ini. Kebutuhan diklasifikasikan menjadi dua bagian yaitu kebutuhan fungsional serta kebutuhan non-fungsional, dimana kebutuhan fungsional ditujukan untuk menjawab permasalahan yang diangkat pada penelitian, sedangkan kebutuhan non-fungsional mencakup kebutuhan akan perangkat pendukung dalam menjalankan penelitian.

3.2.1 Kebutuhan Fungsional

Kebutuhan fungsional menjabarkan tentang hal-hal yang mampu dilakukan oleh sistem, informasi apa saja yang harus tersedia, serta apa saja yang mampu dihasilkan oleh sistem. Berikut merupkan kebutuhan fungsional dari sistem ini:

1. Sistem dapat melakukan identifikasi atas informasi terhadap paket masuk yang menggunakan protokol IPv6.

2. Jaringan SDN yang meliputi:
    - *Switch* (menjalankan protokol OpenFlow serta forwarding paket antar *host*)
    - Kontroler (menjalankan *framework* Ryu serta mendukung mekanisme *Stateful Packet Inspection*)
    - *Host* (memiliki antarmuka jaringan IPv6 serta mampu menerima dan mengirim paket)
3. *Duplicate address detection Attack Simulator* yang mampu menjalankan serangan *DAD DoS* pada jaringan OpenFlow.
4. Sistem dapat menyimpan *state*/informasi dari koneksi yang terbentuk melalui protokol IPv6.
5. Sistem dapat menentukan aksi yang tepat apakah paket tersebut dapat diteruskan (*forward*) atau dibuang (*drop)* berdasarkan pengkondisian sistem dan penentuan *flow*.

3.2.2 Kebutuhan Non-Fungsional

Kebutuhan non-fungsional mencakup kebutuhan *hardware* maupun *software* yang mendukung perancangan, pembangunan sistem dan pengujian penelitian. Berikut merupakan kebutuhan non-fungsional dari sistem ini:

- OpenFlow 1.3: Sistem *Stateful Packet Inspection* yang sebelumnya dikembangkan untuk protokol OpenFlow 1.3)
- Linux Ubuntu: sebagai sistem operasi yang mendukung Mininet serta THC-IPv6 Toolkit.
- Mininet: sebagai simulator pembangun topologi SDN.
- Ryu*:* sebagai SDN *Controller* yang mendukung sistem *Stateful Packet Inspection* sebelumnya.
- Ostinato Traffic Generator: sebagai packet monitoring yang terintegrasi dengan Wireshark.
- THC-IPv6 *Toolkit*: sebagai *assessment* dan *penetration tool* pada jaringan yang digunkan untuk menguji sistem terkait serangan *DAD DoS.*

3.3 Perancangan

Perancangan menjabarkan tahapan perencanaan dan perancangan dasar yang dibutuhkan sistem berdasarkan analisis kebutuhan yang telah dijabarkan. Untuk memenuhi kebutuhan pengembangan *Stateful Packet Inspection* pada jaringan OpenFlow*,* tahapan perancangan yang dilakukan adalah: perancangan topologi jaringan, dimana perancangan ini menentukan *switch*, *host* dan kontroler dari jaringan yang digunakan pada simulator Mininet. Perancangan serangan *duplicate*

address detection DoS, dimana perancangan ini merencanakan skema serangan pada proses *duplicate address detection* dengan memanfaatkan *tool* yang mendukung. Perancangan pengembangan *Stateful Packet Inspection*, dimana perancangan ini merencanakan mekanisme dari pengembangan paket berdasarkan kebutuhan yang diinginkan. Gambaran alur kerja sistem melalui perancangan diagram alir sistem yang ditujukan untuk merencanakan alur kerja sistem secara sistematis.

3.4 Implementasi

Implementasi mencakup tahapan yang dijalakan untuk membangun sistem yang telah dirancang sesuai definisi serta batasan dalam membangun sistem. Berdasarkan penjabaran pada bab perancangan, implementasi yang akan dilakukan secara bertahap mencakup:

1. Instalasi Kontroler Ryu sebagai *framework* kontroler SDN
2. Konfigurasi antarmuka jaringan IPv6 pada *host* yang digunakan.
3. Instalasi Ostinato sebagai pemantau paket pada *host* yang terhubung.
4. Instalasi THC-IPv6 Toolkit sebagai simulator dalam menjalankan serangan *DAD DoS.*
5. Implementasi pengembangan *connection tracking* pada *Stateful Packet Inspection*.
6. Implementasi pengembanagn *ping handle*, *neighbor advertisement handle*, serta *neighbor solicitation handle* pada sistem yang menggunakan protokol IPv6.
7. Implementasi algoritme modifikasi *flow table* berdasarkan paket IPv6 yang masuk.

3.5 Pengujian

Pengujian dilakasanakan dengan melakukan pengujian terkait keberhasilan sistem dalam mendeteksi serta menentukan *action* pada paket yang menggunakan IPv6 pada jaringan OpenFlow. Pengujian pertama dilakukan dengan menjalankan *Stateful Packet Inspection* yang belum dikembangkan dengan dilakukan pengujian serangan *DAD DoS* untuk melihat bagaimana hasil uji serangan terhadap sistem yang belum dikembangkan. Pengujian kedua adalah pengujian kinerja dari pengembangan *Stateful Packet Inspection* dengan menjalankan pengujian komunikasi ping antar *host*, pengujian *duplicate address detection* normal, pengujian *duplicate address detection* DoS dengan satu penyerang serta pengujian *duplicate address detection* DoS dengan banyak penyerang.

3.6 Analisis Hasil

Analisis hasil dilakukan setelah proses pengujian dilakukan. Pada bab ini dijelaskan tentang penjabaran data dan analisis kinerja dari pengembangan *Stateful Packet*

Inspection yang telah didapat pada proses pengujian. Hasil dan pembahasan tersebut melingkupi hasil uji fungsionalitas sistem berdasarkan skema pengujian yang telah dilakukan.

3.7 Kesimpulan

Pengambilan kesimpulan dilaksanakan berdasarkan hasil dari respon sistem terhadap parameter pengujian dari pengembangan *Stateful Packet Inspection* untuk *duplicate address detection DoS* pada OpenFlow yang telah dilakukan. Berdasarkan hasil pengujian, dapat ditentukan apakah pengembangan *Stateful Packet Inspection* tersebut dapat menyelesaikan permasalahan serta kebutuhan yang telah ditentukan.

repository.ub.ac.id

UNIVERSITAS BRAWIJAYA

BAB 4
PERANCANGAN

Sebelum dilakukan implementasi dan pengujian, perlu dilakukan perancangan agar proses implementasi dan pengujian dapat dilakukan secara runtut serta efesien. Pada bab ini perancangan dilakukan untuk melakukan perencanaan terhadap sistem yang dibangun berdasarkan kebutuhan yang telah dispesifikasikan.

4.1 Topologi Jaringan

Dalam membangun topologi yang akan digunakan dalam pengujian sistem, terdapat satu cara untuk melakukan membangun topologi di Mininet yaitu dengan memanfaatkan *GUI* yang dikenal sebagai Miniedit, yang akan dibahas di bagian ini. Berikut ini langkah-langkah membangun topologi di Miniedit, yaitu sebagai berikut.

1. Masuk ke dalam direktori Mininet/examples, lalu jalankan miniedit di *terminal* dengan mengetik perintah:

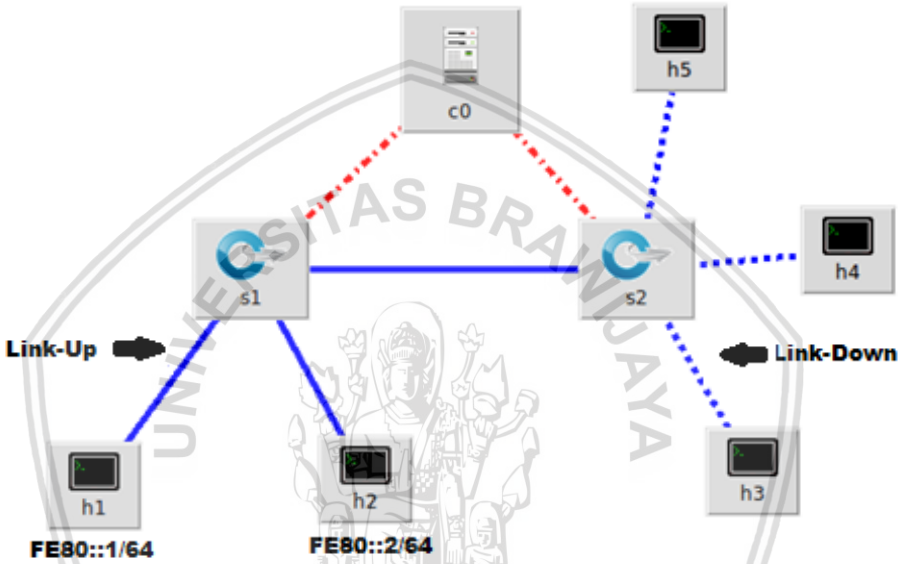
   "$ sudo Python miniedit.py"

2. Membangun topologi jaringan dengan memanfaatkan komponen yang disediakan Mininet. Pada tabel 4.1 (Lazuardi, 2016), merupakan daftar komponen yang disediakan oleh Mininet.

Tabel 4.1 Daftar komponen Mininet

No	Ikon	Nama	Fungsi
1		Kursor	Memindahkan dan memilih komponen.
2		*Host/end Service*	Bertindak sebagai *host/end device* yang menggunakan layanan jaringan.
3		OpenFlow *switch*	Bertindak sebagai perangkat jaringan OpenFlow yang bertindak sebagai *data plane* SDN.
4		*Switch* tradisional	Bertindak sebagai *switch* tradisional yang meneruskan paket menggunakan pemetaan alamat MAC berdasarkan letak *port* bersambungnya.
5		*Router* tradisional	Bertindak sebagai *router* tradisional yang mengambil keputusan penerusan paket berdasarkan protokol *routing*.

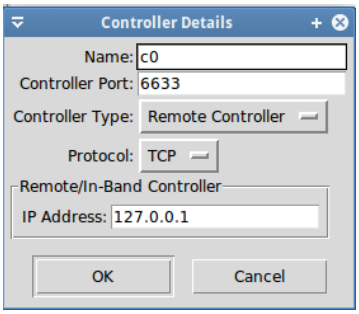
6		*Link*	Menghubungkan antara satu komponen dengan yang lainnya.
7		*Controller*	Bertindak sebagai *Controller* SDN yang diakses oleh Mininet pada *port* tertentu.



Gambar 4.1 Topologi jaringan pada arsitektur SDN

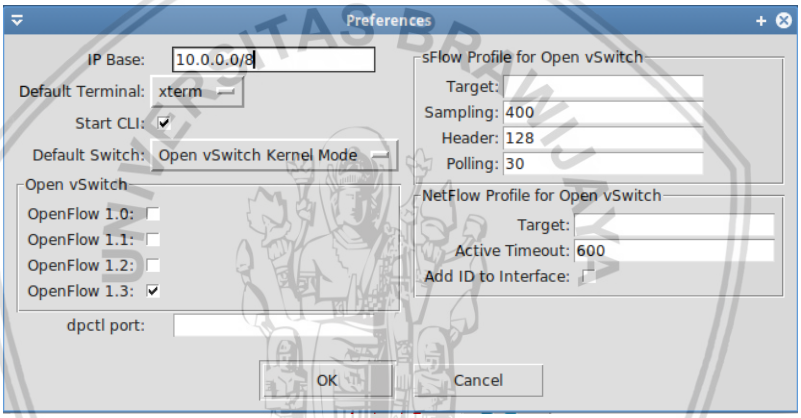
Topologi yang digunakan terdiri atas 5 virtual *host*, 1 *controller* dan 2 *switch* pada arsitektur SDN. Perancangan topologi dibuat secara linier yaitu dengan cara menghubungkan tiap *host* secara langsung pada *switch*. Bentuk topologi tersebut dipilih karena mampu mendeskripsikan pengujian dari *Stateful Packet Inspection* secara sederhana.

3. Konfigurasi *controller* pada topologi sesuai dengan gambar 4.3 dengan cara klik kanan pada entitas *controller* (c0) serta memilih pilihan "*properties"* serta opsi *"Controller Type"* diubah menjadi "*Remote Controller*". Hal tersebut dilaksanakan agar Mininet dapat terhubung dengan program *controller* yang nantinya dibangun.



Gambar 4.2 Pengaturan *controller* di Mininet

4. Melakukan konfigurasi terhadap *preferences* dari Mininet untuk menjalankan fungsi *CLI* Mininet serta menentukan versi OpenFlow yang digunakan sesuai dengan gambar 4.4. Hal tersebut dilaksanakan dengan memilih menu "*Edit*" kemudian memilih "*Preferences*".

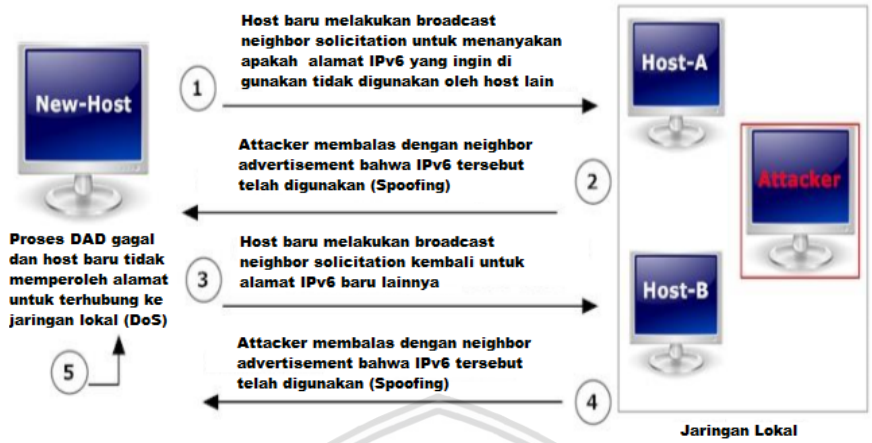


Gambar 4.3 Pengaturan *preferences* di Mininet

5. Menjalankan simulasi topologi yang telah dibangun di Mininet dengan menekan tombol "*Run*" yang tersedia.

4.2 Perancangan Serangan *Duplicate Address Detection* DoS

Dalam perancangan ini, serangan *duplicate address detection DoS* pada sistem dilakukan dengan mengunakan *tool* THC-IPv6 toolkit yang dapat berjalan di sistem operasi linux. Metode serangan yang digunakan adalah *spoofing*, dimana penyerang menyamar sebagai pengguna IPv6 yang ingin digunakan *host* baru dan melakukan modifikasi pada paket *neighbor advertisement* untuk membalas paket *neighbor solicitation* yang dikirim oleh *host* baru.



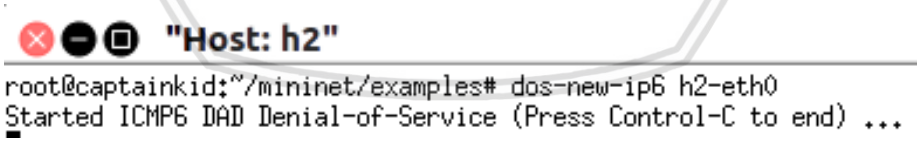
Gambar 4.4 *Duplicate address detection DoS*

Dalam melakukan serangan terhadap sistem dengan metode tersebut, dijalankan *tool* dengan perintah sebagai berikut:

"$ sudo dos-new-ipv6 eth0 "

- dos-new-IPv6 merupakan perintah untuk menjalankan *DAD DoS*, dimana tool akan membalas setiap *neighbor solicitation* dari *host* baru dengan *neighbor advertisement* yang telah di modifikasi (*spoofing*).
- eth0 merupakan *network interface* yang digunakan *attacker* untuk komunikasi pada jaringan lokal.

Untuk menentukan apakah tool dapat menjalankan serangan seperti yang diinginkan, maka dilakukan percobaan serangan dengan menjalankan program simple_*switch*_13.py yang disediakan oleh kontroler Ryu pada topologi yang telah dibangun sebelumnya.



Gambar 4.5 *DAD DoS*

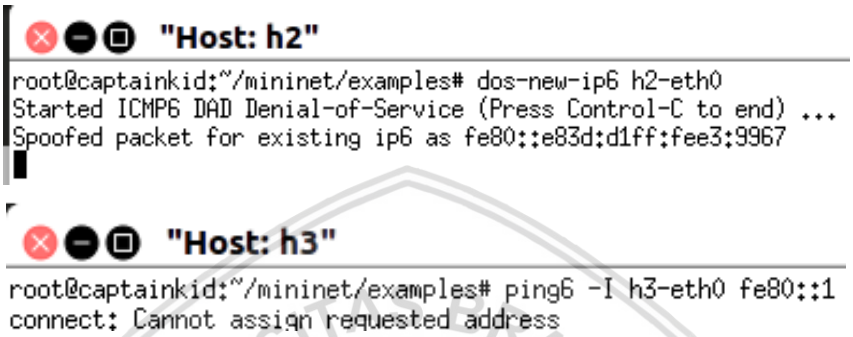
Dalam melakukan serangan terhadap sistem dengan metode tersebut, dijalankan *tool* dengan perintah sebagai berikut:

"$ sudo dos-new-ipv6 eth0 "

- dos-new-IPv6 merupakan perintah untuk menjalankan *DAD DoS*, dimana tool akan membalas setiap *neighbor solicitation* dari *host* baru dengan *neighbor advertisement* yang telah di modifikasi (*spoofing*).

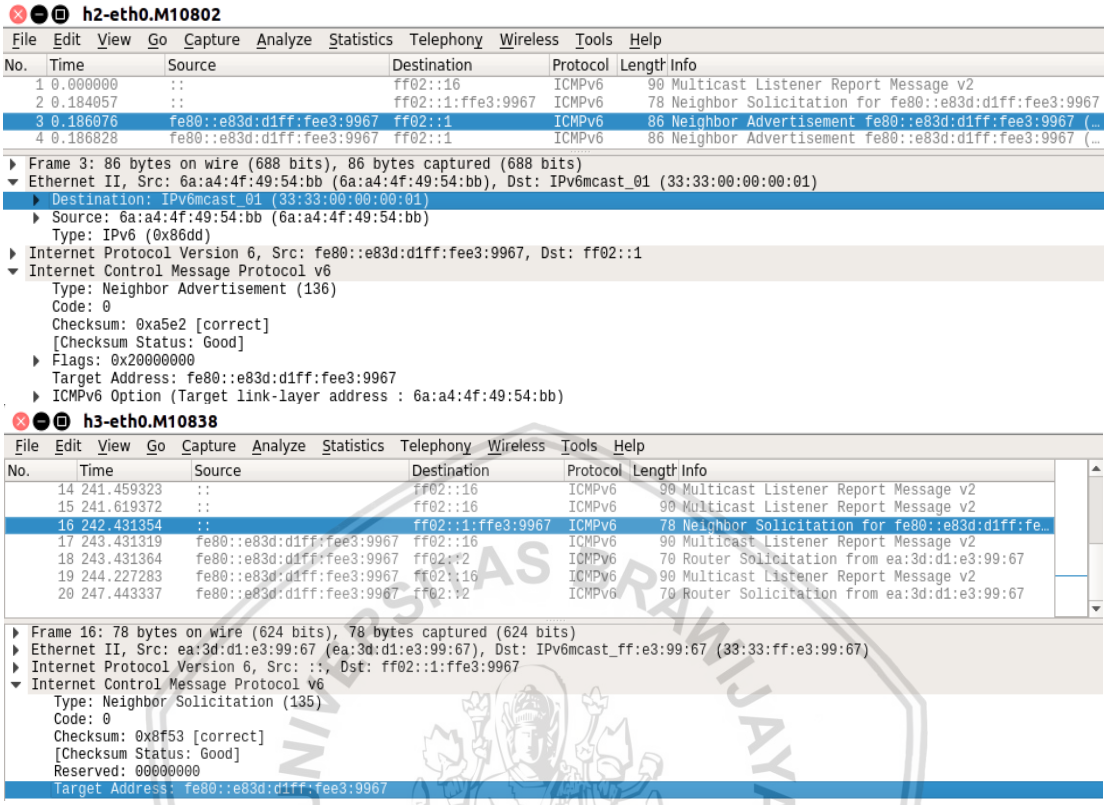
- eth0 merupakan *network interface* yang digunakan *attacker* untuk komunikasi pada jaringan lokal.

Untuk menentukan apakah tool dapat menjalankan serangan seperti yang diinginkan, maka dilakukan percobaan serangan dengan menjalankan program simple_*switch*_13.py yang disediakan oleh kontroler Ryu pada topologi yang telah dibangun sebelumnya.



Gambar 4.6 Output penyerangan dan hasil serangan

Gambar 4.6 menunjukkan bahwa penyerang (h2) berhasil melakukan *spoofing* terhadap *neighbor solicitation* yang dikirim oleh *host* baru (h3) yang mengakibatkan *host* h3 tidak melakukan komunikasi (*ping* dari h3 ke fe80::1) pada jaringan dikarenakan alamat IPv6 yang ingin digunakan telah terpakai didalam jaringan lokal.

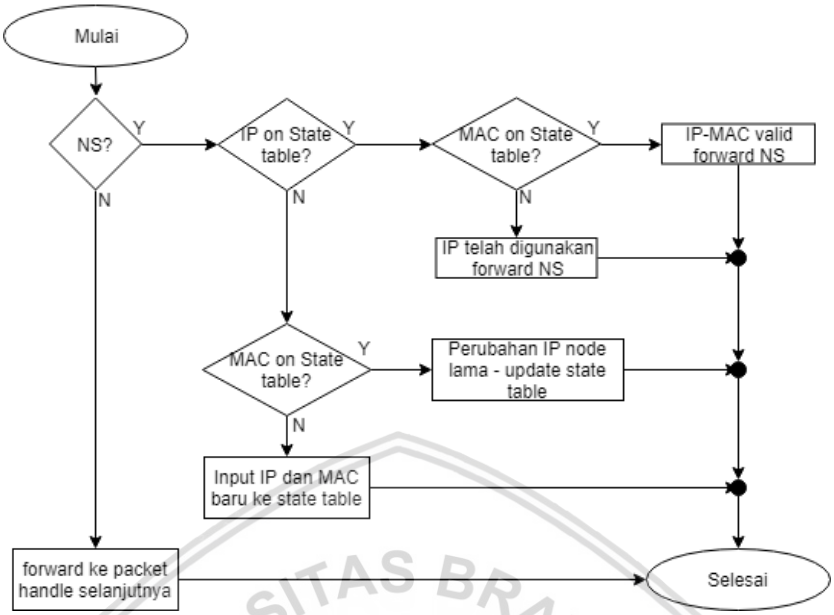


Gambar 4.7 Hasil capture Wireshark pada *attacker*

Gambar 4.7 menunjukkan bahwa *attacker* menerima paket *broadcast* dari *host* baru (*neighbor solicitation*) dimana *host* baru ingin menggunakan alamat fe80::9c54:e9ff:feba:a426. *Attacker* (h2) membalas dengan *neighbor advertisement* yang telah dimodifikasi sehingga *attacker* berpura-pura menjadi *host* yang menggunakan alamat IPv6 yang ingin dipakai oleh *host* baru.

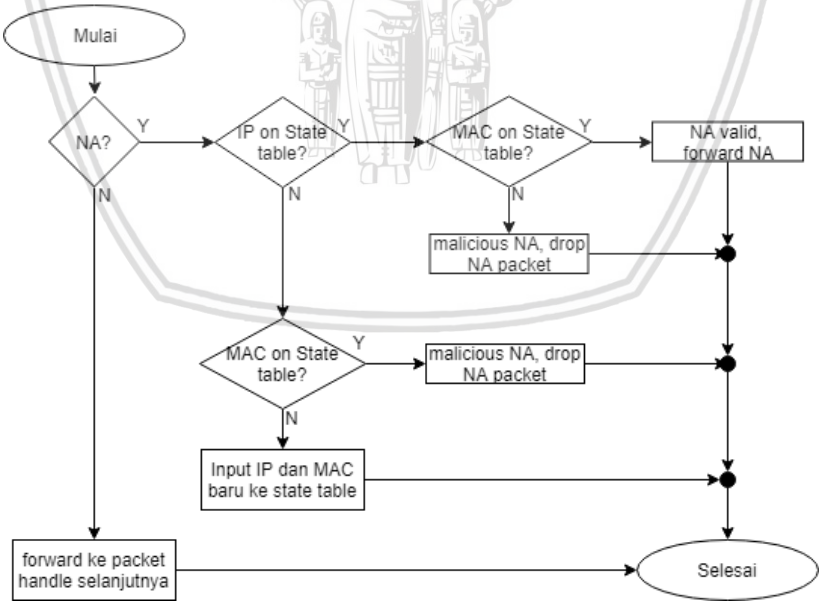
4.3 Perancangan *Stateful Packet Inspection*

Perancangan ini dilakukan guna menggambarkan mekanisme serta tahapan yang digunakan oleh *Stateful Packet Inspection* dalam menangani *DAD* DoS. Mekanisme penanganan paket NDP diterapkan dengan mengadaptasi mekanisme IDS aktif pada penelitian yang dilakukan oleh Ferdous Barbhuiya. Mekanisme ini dikembangkan dengan membangun *packet handler* berdasrkan tipe ICMPv6 dari tiap paket yang masuk. Tiap *packet handler* akan menjalankan aksi berdasarkan informasi yang dibawa oleh paket serta berdasarkan hasil inspeksi terhadap *state table* yang telah ada.



Gambar 4.8 *Neighbor* solicitation handler

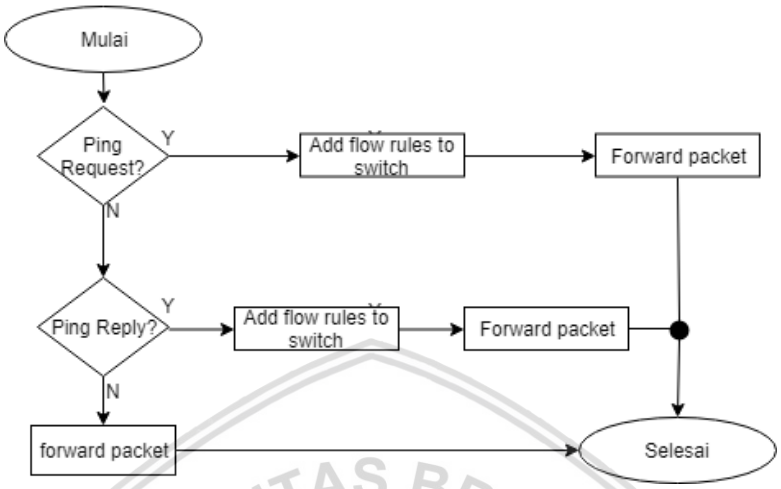
Gambar 4.8 menunjukkan penanganan paket ICMPv6 bertipe NS (*neighbor* solicitation). Packet handler akan melakukan inspeksi terhadap alamat IP dan MAC paket dengan mencocokan informasi tersebut terhadap *state table* yang telah ada sebelumnya. Aksi yang dijalankan sistem akan disesuaikan berdasarkan hasil inspeksi sistem terhadap IP dan MAC dari paket yang masuk.



Gambar 4.9 *Neighbor advertisement* handler

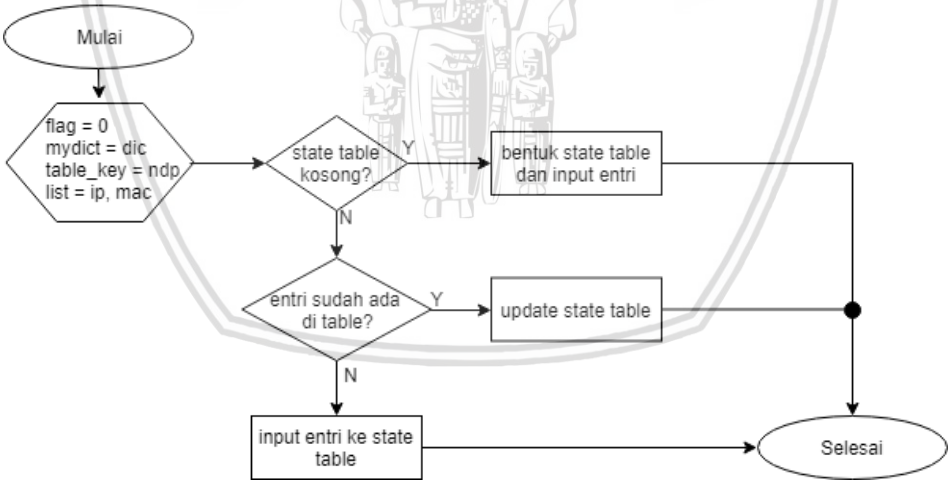
Gambar 4.9 menunjukkan penanganan paket ICMPv6 bertipe NA (*neighbor advertisement*). Packet handler akan melakukan inspeksi terhadap alamat IP dan MAC paket dengan mencocokan informasi tersebut terhadap *state table* yang telah ada

sebelumnya. Aksi yang dijalankan sistem akan disesuaikan berdasarkan hasil inspeksi sistem terhadap IP dan MAC dari paket yang masuk.



Gambar 4.10 Ping handler

Gambar 4.10 menunjukkan penanganan paket ICMPv6 bertipe ping (ping request dan ping reply). Packet handler akan melakukan penambahan *flow rules* dengan pembentukan match *flow rules* berdasarkan informasi dari paket ping yang masuk. Pembentukan *flow rules* hanya dilakukan apabila match rules yang terbentuk bersifat unik (belum terdapat pada flow table *switch*).

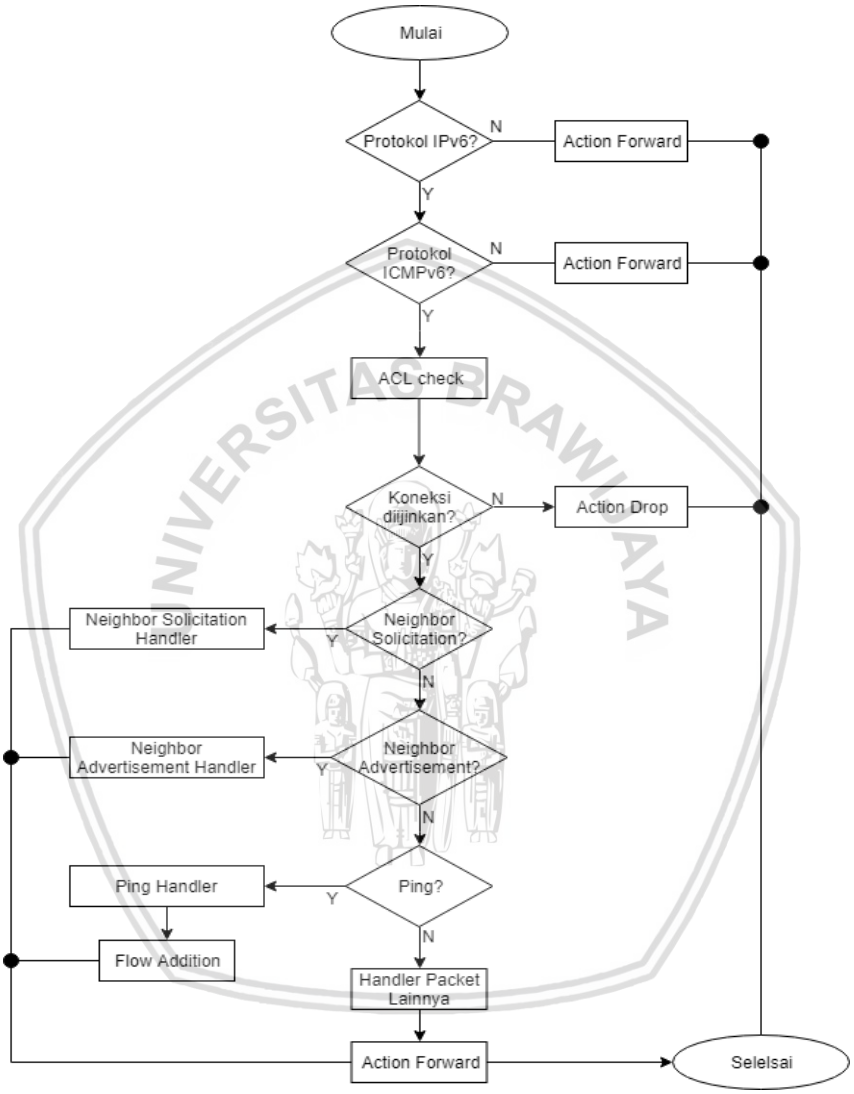


Gambar 4.11 Connection tracking handler

Gambar 4.11 menunjukkan proses pembentukkan state table. Proses ini berguna agar paket inspeksi dapat berjalan secara dinamis (*stateful*) dimana proses diawali dengan pendefinisian variable yang akan digunakan, dilanjutkan dengan melakukan pengecekan apakah *state table* telah terbentuk sebelumnya serta pengecekan entri dari state table yang ingin disimpan.

4.4 Diagram Alir Pengembangan Sistem

Dalam perancangan ini terdapat beberapa tahapan yang harus dilakukan. Berikut ini merupakan gambar diagram alir dari perancangan pengembangan sistem yang dibuat.



Gambar 4.12 Flowchart Pengembangan Sistem

Pertama sistem akan melakukan proses seleksi protokol pada apakah paket yang masuk menggunakan protokol IPv6. Kemudian dilakukan seleksi kembali apakah paket tersebut merupakan paket ICMPv6. Selanjutnya dilakukan proses pengecekan informasi koneksi yang masuk terhadap ACL yang tersedia. Bila koneksi berhasil melewati proses pengecekan ACL, maka akan seleksi terhadap tipe paket yang masuk untuk diteruskan kepada handle paket yang sesuai dengan tipenya.

BAB 5
IMPLEMENTASI

Pada bab ini akan dijelaskan mengenai implementasi dari *Stateful Packet Inspection* IPv6, dimana implementasi mencakup pembentuk pengkondisioan paket berdasarkan protokol, pengkondisian paket *neighbor solicitation*, pengkondisian paket dengan *neighbor advertisement*, pengkondisian paket ICMPv6 lainnya (*ping reply, ping request, dan sebagainya*), serta implementasi *connection tracking*. Implementasi dilakukan dengan memodifikasi kontroler Ryu dan berjalan pada jaringan OpenFlow.

5.1 Implementasi Lingkungan Pengujian

5.1.1 Instalasi Ryu *Controller*

Berikut merupakan tahapan instalasi *API* Ryu sebagai kontroler pada SDN dengan memanfaatkan *source code* pada github. Memasang git yang digunakan untuk penyalinan *source code* dari github resmi milik Ryu. Instalasi dilakukan di terminal dengan menjalankan perintah "sudo apt-get install git". Kemudian, salin *source code* Ryu yang disediakan github dengan mejalankan perintah "sudo git clone git://github.com/osrg/ryu.git". Lakukan instalasi *file setup* pada *folder* Ryu yang telah disalin dari github dengan menjalakan perintah "sudo ./setup.py install".

```
adk427@n4:~/ryu/ryu/app$ ryu-manager --verbose ryu.app.example_switch_13
loading app ryu.app.example_switch_13
loading app ryu.controller.ofp_handler
instantiating app ryu.app.example_switch_13 of ExampleSwitch13
instantiating app ryu.controller.ofp_handler of OFPHandler
BRICK ExampleSwitch13
  CONSUMES EventOFPPacketIn
  CONSUMES EventOFPSwitchFeatures
BRICK ofp_event
  PROVIDES EventOFPPacketIn TO {'ExampleSwitch13': set(['main'])}
  PROVIDES EventOFPSwitchFeatures TO {'ExampleSwitch13': set(['config'])}
  CONSUMES EventOFPErrorMsg
  CONSUMES EventOFPPortDescStatsReply
  CONSUMES EventOFPEchoRequest
  CONSUMES EventOFPHello
  CONSUMES EventOFPEchoReply
  CONSUMES EventOFPPortStatus
  CONSUMES EventOFPSwitchFeatures
connected socket:<eventlet.greenio.base.GreenSocket object at 0x7ffb5d39b6d0> address:('127.0.0.1', 50331)
hello ev <ryu.controller.ofp_event.EventOFPHello object at 0x7ffb5d39bf50>
move onto config mode
EVENT ofp_event->ExampleSwitch13 EventOFPSwitchFeatures
switch features ev version=0x4,msg_type=0x6,msg_len=0x20,xid=0x4973b8c8,OFPSwitchFeatures(auxiliary_id=0,capabilities=71,datapath_id=1
,n_buffers=256,n_tables=254)
move onto main mode
```

Gambar 5.1 Pengaturan *preferences* di Mininet

Setelah proses instalasi selesai, lakukan uji coba aplikasi Ryu dengan menjalankan contoh program yang disediakan dengan menjalankan perintah "ryu-manager" seperti pada gambar 5.1. Ryu menyediakan banyak varian framework yang dapat disesuaikan untuk pengembangan aplikasi pada jaringan OpenFlow.

5.1.2 Konfigurasi *Network Interface*

Dalam melakukan implemetasi *Stateful Packet Inspection* untuk IPv6, *host* dari topologi yang digunakan harus mendukung *interface* IPv6. Konfigurasi alamat IPv6 pada *host* 1 dan *host* 2 dilakukan guna mempermudah proses pengujian sistem. Berikut ini adalah langkah-langkah dalam menambahkan *interface* IPv6 pada *host* di Mininet. Buka CLI Mininet yang ada setelah menjalankan topologi di Mininet.

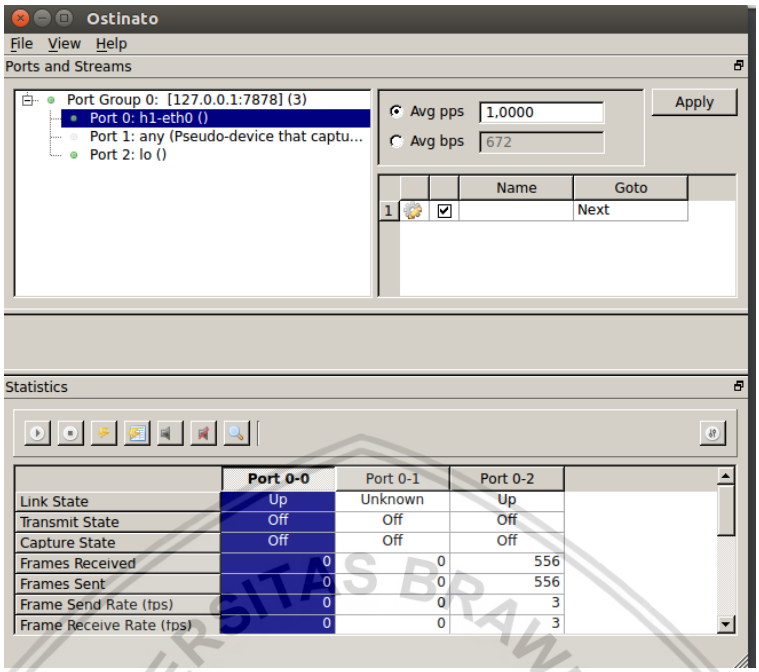
```
*** Starting CLI:
mininet> h1 ifconfig h1-eth0 inet6 add fe80::1/64
mininet> h2 ifconfig h2-eth0 inet6 add fe80::2/64
```

Gambar 5.2 Pengaturan *preferences* di Mininet

Untuk menambahkan alamat IPv6 pada *host* 1, lakukan perintah berikut "h1 ifconfig h1-eth0 inet6 add fe80::1/64". Untuk menambahkan alamat IPv6 pada *host* 2, lakukan perintah berikut "h2 ifconfig h2-eth0 inet6 add fe80::2/64"

5.1.3 Instalasi Ostinato

Ostinato adalah generator dan penganalisis paket pada trafik jaringan dengan fitur *GUI* yang ramah yang dilengkapi *API* Python yang kuat untuk otomasi uji jaringan. Pada peneliitan ini, ostinato digunakan untuk mengirim paket modifikasi guna melakukan pengujian terhadap sistem serta melakukan *capture* pada *interface* jaringan. Berikut adalah cara instalasi ostinato. *Install* melalui terminal dengan menjalankan perintah "sudo apt-get install ostinato" pada terminal. Jalankan program ostinato dengan perintah "sudo ostinato" pada terminal.



Gambar 5.3 Tampilan Ostinato

Gambar 5.2 merupakan tampilan *GUI* dari Ostinato yang berhasil terpasang pada sistem operasi setelah menjalankan "sudo ostinato" pada terminal. Port yang akan dipantau melalui Ostinato adalah *port* 0 yang merupakan *default port number* dari *ethernet* pada aplikasi Ostinato.

5.1.4 Instalasi THC-IPv6 *Toolkit*

IPv6 *toolkit* merupakan *assessment* dan *penetration tool* pada jaringan dengan memanfaatkan protokol IPv6. Pada penelitian ini, IPv6 *toolkit* digunakan sebagai alat yang mengeksekusi serangan *DAD DoS* terhadap sistem. Berikut merupakan tahapan dalam instalasi *THC*-IPv6 *toolkit*:

1. Unduh *package* IPv6 *toolkit* terbaru melalui *link* dibawah ini:

   "https://github.com/vanhauser-thc/thc-ipv6.git"

2. Ekstrak *package* yang telah diunduh dengan menjalankan perintah:

   "sudo tar -xzf thc-ipv6.tar.gz"

3. Masuk ke direktori file yang telah di ekstrak sebelumnya, lalu jalankan perintah:

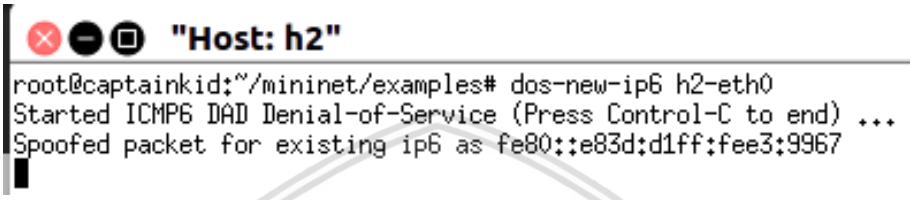
   "sudo make all"

4. Selanjutnya, jalankan perintah:

   "sudo make install"

5. Jalankan program dengan mencoba salah satu *tool* yang tersedia dengan menjalankan perintah:

   "sudo dos-new-ipv6"



Gambar 5.4 THC-IPv6 Toolkit

Gambar 5.4 merupakan keluaran dari salah satu fungsi program THC-IPv6 Toolkit, yaitu *neighbor advertisement DoS.* THC-IPv6 Toolkit memiliki berbagai macam fungsi dari IPv6 *peneteration tool* yang dapat digunakan sesuai keinginan pengguna.

5.2 Implementasi Pengembangan SIstem

5.2.1 Implementasi *Connection Tracking*

Implementasi ini dilakukan untuk membangun program yang dapat menyimpan *connection state* yang aktif. Implemtasi *connection tracking* menggunakan variable *dictionary* pada bahasa Python, dimana informasi yang disimpan ke dalam *dictionary* adalah alamat IPv6 serta alamat *MAC*.

Source code: connection tracking

```
1     def conn_track_dict(self,dic,ndp,s_ip,mac):
2         flag = 0
3         mydict = dic
4         table_key = str(ndp)
5         list1 = [s_ip, mac]
6         listobj = []
7
8         # cek bila state table kosong (pembentukan awal)
9         if(mydict.has_key(table_key) is False):
10            key = table_key
11            tup = tuple(list1)
12            listobj.append(tup)
13            tup = tuple(listobj)
14            mydict[key] = tup
15            flag = 1
16
17        # Check bila entry yang masuk sudah ada di state table
18        elif (mydict.has_key(table_key) is True):
19            for x in list(mydict[table_key]):
20                if (list1 == list(x)):
```

```
21                      flag = 1
22                      break
23
24          # check untuk flow unik
25          if(flag != 1):
26              key = table_key
27              dst = mydict[key]
28              dst = list(dst)
39              dst.append(tuple(list1))
30              tup = tuple(dst)
31              mydict[key] = tup
32
33          return mydict
```

Berdasarkan *source code* diatas maka dapat diketahui alur *source code* dari *connection tracking* tersebut. Berikut penjelasan *source code* diatas:

- Baris 1, merupakan nama *method* dari *connection tracking* dengan membawa parameter *self* (merupakan *dictionary state table* yang ada pada *main program*), dic, ndp (*key* dari *dictionary*), alamat IPv6 serta alamat MAC.
- Baris 2 merupakan inisiasi variable "flag" dengan nilai 0 yang digunakan untuk parameter pengkondisian (*trigger*).
- Baris 3 merupakan variable "mydic" dengan nilai *dictionary* yang dibawa pada saat *method* dijalankan.
- Baris 4, merupakan pembuatan variable "*table*_key" yang nanti digunakan sebagai *key* pada *dictionary* dengan nilai *string* dari variable "ndp".
- Baris 5 merupakan pembentukan obyek *list* dengan nama "list1" yang nilai didalamnya adalah alamat IPv6 serta alamat MAC.
- Baris 6 merupakan pembentukan list "listobj" yang digunakan sebagai penampung *match* dari *state table.*
- Baris 8-15, merupakan pembuatan *dictionary* (*state table*) dengan *key* baru bila nilai "ndp" tidak ditemukan pada *dictionary* yang sudah ada. Mekanisme ini dijalankan bila nilai *key* pada dictionary belum ditemukan.
- Baris 17-22, merupakan pengecekan apabila nilai *state* yang ingin dimasukkan telah terdapat pada *state table* yang sudah ada, sehingga tidak perlu ditambahkan ke *dictionary state table*. Mekanisme ini dijalankan apabila nilai *key* pada *dictionary* telah ditemukan (sesuai), sehingga nilai list langsung dimasukkan sebagai entri baru pada *dictionary* dengan *key* tersebut.
- Baris 24-31, merupakan *source* yang dijalankan bila entri yang ingin dimasukkan ke dalam *dictionary state table* terdapat kesamaan *key* (ndp) pada *dictionary* dan isi *list* dari entri berbeda dari yang sudah ada pada *dictionary state table*. Mekanisme ini dijalankan apabila nilai variable flag (trigger) tidak

sama dengan 1, yang menandakan pembentukan entri *state table* yang unik (pembentukan awal *dictionary*).

- Baris 33, merupakan fungsi *return* untuk mengembalikan nilai akhir dari *dictionary state table* ke program utama.

5.2.2 Implementasi Packet Inspection

Implementasi ini dilakukan untuk membangun program utama dimana *Stateful Packet Inspection* berjalan. Dalam pelaksanaannya, inti dari implementasi ini adalah membangun *packet handle* yang sesuai dengan protokol yang digunakan, tipe dari ICMPv6 yang dipakai serta informasi dari koneksi (IP asal dan tujuan, *port* asal dan tujuan, dan lainnya).

5.2.2.1 *Implementasi Ping Handle*

Implementasi ini dilakukan untuk membangun program yang dapat menangani paket ICMPv6 dengan tipe *echo reply* dan *echo request*. Pada pelaksanaannya, algoritme ini dibangun untuk melakukan pengecekan paket berdasarkan yang dibawa. Algoritme ini juga dirancang untuk melakukan pembentukan *flow rules* yang akan disimpan ke dalam *switch*.

Source code: *Ping Handle*

```
1         if (out_port != ofproto.OFPP_FLOOD) and (ethtype ==
2  ETH_TYPE_IPV6):
3                  ipo = pkt.get_protocols(IPv6.IPv6)[0]
4                  if (ipo.nxt == IPPROTO_ICMPV6):
5                      icmpob = pkt.get_protocol(icmpv6.icmpv6)
6                      flag1 = 0
7                      # Cek bila paket PING
8                      if (icmpob.type_ == 128):
9
10                         flag1 = 1
11                         actions_default = action_fwd_to_out_port
12                         self.logger.info("%s  ->  %s : ECHO REQUEST
13 ALLOWED" % (ipo.src, ipo.dst))
14                         self.flow.add_flow(datapath=datapath,
15 actions=actions_default, priority=1001, in_port=in_port,
16                                                 eth_type=ETH_TYPE_IPV6,
17 ip_proto=IPPROTO_ICMPV6, icmpv6_type=128,
18                                                 IPv6_src=ipo.src,
19 IPv6_dst=ipo.dst)
20
21                     # Otherwise, PONG...!
22                     elif (icmpob.type_ == 129):
23                         flag1 = 1
24                         actions_default = action_fwd_to_out_port
25                         self.logger.info("%s  ->  %s : ECHO REPLY
26 ALLOWED" % (ipo.src, ipo.dst))
27                         self.flow.add_flow(datapath=datapath,
28 actions=actions_default, priority=1001,
29                                                 in_port=in_port,
30
```

```
31                                                  eth_type=ETH_TYPE_IPV6,
32  ip_proto=IPPROTO_ICMPV6, icmpv6_type=129,
33                                                  IPv6_src=ipo.src,
    IPv6_dst=ipo.dst)
```

Berdasarkan *source code* diatas maka dapat diketahui alur *source code* dari *ping handle* tersebut. Berikut penjelasan *source code* diatas:

- Baris 1-2, merupakan kondisi utama yang dijalankan bila protokol dari paket yang masuk merupakan IPv6 serta paket tersebut memiliki satu alamat tujuan(unicast) serta pembuatan obyek "ipo" sebagai obyek inisialisai penggunaan protokol IPv6.
- Baris 3, merupakan pengkondisian selanjutnya dimana kondisi dinyatakan benar bila paket yang masuk merupakan ICMPv6.
- Baris 4 merupakan pembentukan obyek "icmpob" sebagai obyek inisialisasi penggunaan protokol ICMPv6.
- Baris 5 merupakan inisialisasi variable flag sebagai trigger untuk pengkondisian.
- Baris 7, pengkondisian apakah paket yang masuk merupakan ICMPv6 dengan type 128 (*echo request*).
- Baris 9-10, mengubah nilai variable "flag" menjadi 1 serta melakukan *action forward* paket ke *switch*.
- Baris 13-17, merupakan pembentukan *flow rules* berdasarkan hasil *filtering* pada informasi paket yang masuk. Pembentukan match rules dari *flow rules* dibentuk berdasarkan informasi dari koneksi yang terhubung.
- Baris 20, merupakan pengkondisian apabila paket merupakan ICMPv6 dengan tipe 129 (*echo reply*).
- Baris 21, memberikan nilai "flag" menjadi 1.
- Baris 22 merupakan mekanisme *forwarding* paket ke *switch*.
- Baris 23 merupakan tampilan keluaran berupa teks tertulis "ECHO REPLY ALLOWED" dan ikutin nilai dari alamat IP tujuan dan alamat IP asal.
- Baris 25-33, merupakan pembentukan *flow rules* berdasarkan hasil *filtering* paket yang masuk. Pembentukan match rules dari *flow rules* dibentuk berdsarkan informasi dari koneksi yang terhubung.

5.2.2.2 *Implementasi Neighbor Solicitation Handle*

Implementasi ini dilakukan untuk membangun program yang dapat menangani paket ICMPv6 dengan tipe 135 (*neighbor solicitation*). Pada pelaksanaannya, algoritme ini dibangun untuk melakukan pengecekan informasi paket berdasarkan *state table* yang telah disimpan pada *dictionary* serta algoritme ini juga dirancang melakukan pengecekan validasi paket *neighbor solicitation*, apakah informasi dari paket tersebut tersedia pada *state table* yang ada.

Source code: Neighbor Solicitation Handle

```
1   elif (icmpob.type_ == 135):
2     self.logger.info("%s  ->  %s : Neighbour Solicitation" % (ipo.src,
3
4     trig = 1
5     if ('135' in self.icmp_conn_track) and ipo.src is not None:
6         print ' '
7         for i in self.icmp_conn_track:
8             print i, self.icmp_conn_track[i]
9         temp = self.icmp_conn_track.get('135')
10        for i in range(0, len(temp)):
11            if ipo.src == "::":
12                if (temp[i][0] == icmpob.data.dst) and (temp[i][1] ==
13
14                    print 'IPv6 address valid/telah tersimpan'
15                    actions_default = action_fwd_to_out_port
16                    trig = 2
17                    break
18                elif (temp[i][0] == icmpob.data.dst) and (temp[i][1] !=
19
20                    print 'IP telah digunakan di jaringan lokal'
21                    actions_default = action_fwd_to_out_port
22                    trig = 2
23                    break
24                elif (temp[i][0] != icmpob.data.dst) and (temp[i][1] ==
25
26                    print 'input alamat IPv6 baru pada node lama'
27                    self.icmp_conn_track.pop('135', None)
28                    for j in range(0, len(temp)):
29                        if (temp[j][1] != eth.src):
30                            self.icmp_conn_track =
31  rack_dict(
32                                self.icmp_conn_track, '135',
33  j][1])
34                    self.icmp_conn_track = self.track.conn_track_dict(
35                        self.icmp_conn_track, '135', icmpob.data.dst,
36
37                    actions_default = action_fwd_to_out_port
38                    trig = 2
39                    for i in self.icmp_conn_track:
40                        print i, self.icmp_conn_track[i]
41                    break
42            else:
43                if (temp[i][0] == ipo.src) and (temp[i][1] == eth.src):
44                    print 'IPv6 address valid/telah tersimpan'
45                    actions_default = action_fwd_to_out_port
46                    trig = 2
47                    break
48                elif (temp[i][0] == ipo.src) and (temp[i][1] !=
49
50                    print 'IP telah digunakan di jaringan lokal'
51                    actions_default = action_fwd_to_out_port
52                    trig = 2
53                    break
54                elif (temp[i][0] != ipo.src) and (temp[i][1] ==
55
```

```
56                      print 'input alamat IPv6 baru pada node lama'
57                      self.icmp_conn_track.pop('135', None)
58                      for j in range(0, len(temp)):
59                          if (temp[j][1] != eth.src):
60                              self.icmp_conn_track =
61  rack_dict(
62                                  self.icmp_conn_track, '135',
63  j][1])
64                      self.icmp_conn_track = self.track.conn_track_dict(
65                          self.icmp_conn_track, '135', ipo.src, eth.src)
66                      actions_default = action_fwd_to_out_port
67                      trig = 2
68                      for i in self.icmp_conn_track:
69                          print i, self.icmp_conn_track[i]
70                      break
71
72     if (trig == 1):
73         print "Input IP + MAC node baru ke state table"
74         if ipo.src is not "::":
75             self.icmp_conn_track =
76  rack_dict(self.icmp_conn_track, '135', ipo.src,
77                                                              eth.src)
78             actions_default = action_fwd_to_out_port
79         else:
80             self.icmp_conn_track =
81  rack_dict(self.icmp_conn_track, '135',
82
83
84                                                              eth.src)
85             actions_default = action_fwd_to_out_port
86         for i in self.icmp_conn_track:
87             print i, self.icmp_conn_track[i]
```

Berdasarkan *source code* diatas maka dapat diketahui alur *source code* dari *neighbor solicitation handle* tersebut. Berikut penjelasan *source code* diatas:

- Baris 1, merupakan pengondisian apabila paket yang masuk merupakan ICMPv6 dengan tipe 135 (*neighbor solicitation*).
- Baris 2 merupakan tampilan keluaran berupa teks "*Neighbor* Solicitation" sebagai penanda bahwa paket yang masuk bertipe *neighbor solicitation*.
- Baris 4, merupakan penentuan nilai variable "trig" menjadi 1.
- Baris 5 merupakan pengecekan apakah *state table* telah terbentuk atau tidak.
- Baris 7-8, merupakan tampilan keluaran pada kontroler dari isi *state table* semula.
- Baris 9, menyimpan nilai yang ada pada *state table* ke dalam *list* "temp".
- Baris 10, merupakan mekanisme pengulangan dengan menjalankan perulangan sesuai dari panjangnya *state table*.
- Baris 11, merupakan pengkondisian bila paket *neighbor* solicitation belum memiliki alamat IPv6 tetap (alamat IPv6 masih "::")

- Baris 12-13, merupakan pengkondisian apabila pada *state table* terdapat alamat IPv6 dan alamat MAC dari pengirim paket,
- Baris 14 merupakan tampilan keluaran berupa teks “IPv6 address valid/telah tersimpan”.
- Baris 15 merupakan mekanisme *forwarding* paket ke *switch*.
- Baris 16-17 merupakan pengubahan nilai trig menjadi 2 disertati dengan pengakhiran pengondisian dan perulangan (*break*).
- Baris 18, merupakan pengkondisian apabila pada *state table* terdapat alamat IPv6 namun terdapat perbedaan alamat MAC dari pengirim paket.
- Baris 20 merupakan tampilan keluaran berupa teks 'IP telah digunakan di jaringan lokal'.
- Baris 21-22 merupakan pengubahan nilai trig menjadi 2 disertati dengan pengakhiran pengondisian dan perulangan (*break*).
- Baris 24, merupakan pengkondisian apabila pada *state table* terdapat alamat IPv6 yang berbeda namun terdapat persamaan alamat MAC dari pengirim paket.
- Baris 26 merupakan tampilan keluaran berupa teks 'IP telah digunakan di jaringan lokal'.
- Baris 27 merupakan mekanisme dimana sistem akan membentuk *state table* baru dengan memasukkan alamat IPv6 baru sesuai dengan alamat MAC serta menghapus alamat IPv6 yang lama.
- Baris 28-35 merupakan proses pembentukan entri *state table* yang baru.
- Baris 37-38 merupakan forwarding paket ke *switch* serta pengubahan nilai trig menjadi 2.
- Baris 39-41 merupakan mekanisme untuk melakukan tampilan keluaran dari *state table* terbaru disertati dengan pengakhiran pengondisian dan perulangan (*break*).
- Baris 42-70, merupakan pengkondisian yang serupa seperti baris 12-38, namun penggalan *code* akan dijalankan bila paket *neighbor* solicitation yang masuk telah memilki alamat IPv6 tetap.
- Baris 72, merupakan pengkondisian apabila pada *state table* terdapat tidak ditemukan alamat IPv6 serta alamat MAC dari paket yang masuk.
- Baris 73, merupakan tampilan keluaran berupa teks “Input IP + MAC *node* baru ke *state table”*.
- Baris 74-84 merupakan mekanisme pembentukan entri *state table* unik dimana pengkondisian dibedakan berdasarkan nilai IP asal dari paket yang masuk.
- Baris 85 merupakan mekanisme forwarding paket menuju *switch*.

- Baris 86-87 merupakan mekanisme untuk melakukan tampilan keluaran dari *state table* terbaru.

5.2.2.3 *Implementasi Neighbor Advertisement Handle*

Implementasi ini dilakukan untuk membangun program yang dapat menangani paket ICMPv6 dengan tipe 136 (*neighbor advertisement*). Pada pelaksanaannya, algoritme ini dibangun untuk melakukan pembaharuan nilai *state table* sesuai informasi yang dibawa paket serta algoritme ini juga dirancang melakukan pengecekan validasi paket tersebut, apakah informasi dari paket tersebut tersedia pada *state table* yang ada (*state table* berdasarkan paket sebelumnya).

Source code: neighbor advertisement handle

```
1   elif (icmpob.type_ == 136):
2                               self.logger.info("%s   ->   %s : Neighbour
3   Advertisement" % (ipo.src, ipo.dst))
4                               trig = 1
5                               if '135' in self.icmp_conn_track:
6
7                                   temp = self.icmp_conn_track.get('135')
8                                   print ' '
9                                   for i in range(0, len(temp)):
10                                      if  (temp[i][0]  ==  ipo.src)  and
11  (temp[i][1] == eth.src):
12                                          actions_default             =
13  action_fwd_to_out_port
14                                          trig = 2
15                                          break
16                                      elif  (temp[i][0]  ==  ipo.src)  and
17  (temp[i][1] != eth.src):
18                                          print  'spoofed  NA  detection,
19  drop NA packet'
20                                          actions_default = action_drop
21                                          trig = 2
22                                          break
23
24                                      elif  (temp[i][0]  !=  ipo.src)  and
25  (temp[i][1] == eth.src):
26                                          print  'spoofed  NA  detection,
27  drop NA packet'
28                                          actions_default = action_drop
29                                          trig = 2
30                                          break
31
32                              if (trig == 1):
33                                  print 'NA Unik, input ke state table'
34                                  self.icmp_conn_track                =
35  self.track.conn_track_dict(self.icmp_conn_track, '135', ipo.src,
36
37  eth.src)
38                                  actions_default                     =
39  action_fwd_to_out_port
40                                  for i in self.icmp_conn_track:
41                                      print i, self.icmp_conn_track[i]
```

Berdasarkan *source code* diatas maka dapat diketahui alur *source code* dari *packet handle* tersebut. Berikut penjelasan *source code* diatas:

- Baris 1, merupakan pengondisian apabila paket yang masuk merupakan ICMPv6 dengan tipe 136 (*neighbor advertisement*).
- Baris 2 merupakan tampilan keluaran dari teks "Neighbour *Advertisement*" sebagai penanda bahwa paket yang masuk bertipe *neighbor advertisement.*
- Baris 4, merupakan penentuan nilai variable "trig" menjadi 1.
- Baris 5 merupakan pengecekan apakah *state table* dengan key "135" telah terbentuk atau tidak sebelumnya.
- Baris 7, menyimpan nilai yang ada pada *state table* ke dalam *list* "temp".
- Baris 9-13, merupakan pengkondisian apabila pada *state table* terdapat alamat IPv6 dan alamat MAC dari pengirim paket, serta melakukan *forwarding*.
- Baris 14-18, merupakan pengkondisian apabila pada *state table* terdapat alamat IPv6 namun terdapat perbedaan alamat MAC dari pengirim paket, pada kasus ini sistem mendeteksi apabila telah terjadi *spoofing* sehingga sistem melakukan *drop packet*.
- Baris 20-24, merupakan pengkondisian apabila pada *state table* terdapat alamat IPv6 namun terdapat perbedaan alamat MAC dari pengirim paket, pada kasus ini sistem mendeteksi apabila telah terjadi *spoofing* sehingga sistem melakukan *drop packet*.
- Baris 26-33, merupakan pengkondisian apabila pada *state table* terdapat tidak ditemukan alamat IPv6 serta alamat MAC dari paket yang masuk, sehingga sistem melakukan entri baru dari informasi paket tersebut kedalam *state table*.

5.2.2.4 *Implementasi Packet Handle Lainnya.*

Implementasi ini dilakukan untuk membangun program yang dapat menangani paket yang tidak berhubungan dengan proses *duplicate address detection*. Sistem akan melakukan *forwarding* pada tiap paket tersebut untuk menghidari terjadinya kesalahan dalam komunikasi antar *host*.

Source code: packet handler lainnya

```
1   elif (icmpob.type_ == 137):
2                               self.logger.info("%s  ->  %s : Redirect" %
3   (ipo.src, ipo.dst))
4                           elif (icmpob.type_ == 139):
5                               self.logger.info("%s  ->  %s : Information
6   Query" % (ipo.src, ipo.dst))
7                           elif (icmpob.type_ == 140):
8                               self.logger.info("%s  ->  %s : Information
9   Response" % (ipo.src, ipo.dst))
10                          else:
11                              self.logger.info("%s  ->  %s : (Packet type
12  = %s) " % (ipo.src, ipo.dst, icmpob.type_))
```

Berdasarkan *source code* diatas maka dapat diketahui alur *source code* dari *packet handle* tersebut. Berikut penjelasan *source code* diatas:

- Baris 1-2, merupakan *handle* dari paket ICMPv6 dengan tipe 137 (*redirect*)
- Baris 3-5, merupakan *handle* dari paket ICMPv6 dengan tipe 139 (*information query*)
- Baris 6-8, merupakan *handle* dari paket ICMPv6 dengan tipe 140 (*information response*)
- Baris 9-11, merupakan *handle* dari paket ICMPv6 dengan tipe lainnya

5.2.3 Implementasi Source Code Flow Addition

Implementasi ini dilakukan untuk membangun program yang dapat menambahkan entri *flow* baru ke dalam *flow table*. Implementasi *flow addition* dijalankan bila paket yang masuk sesuai dengan *firewall rules* maupun *state table* yang ada. Penambahan entri *flow* membawa informasi dari paket yang masuk untuk dijadikan nilai "*match*" pada *flow table* (*destination IP*, *source port*, *destination port*, serta protokol yang digunakan).

Source code: flow addition

```
1     def add_flow(self,datapath, priority, match, actions,
2  idle_timeout=1800):
3           ofproto = datapath.ofproto
4           parser = datapath.ofproto_parser
5           inst                                                =
6  [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
7                                                  actions)]
8
9           mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
13                                  match=match,instructions=inst,
14 idle_timeout=idle_timeout)
15          datapath.send_msg(mod)
```

Berdasarkan *source code* diatas maka dapat diketahui alur *source code* dari *flow* addition tersebut. Berikut penjelasan *source code* diatas:

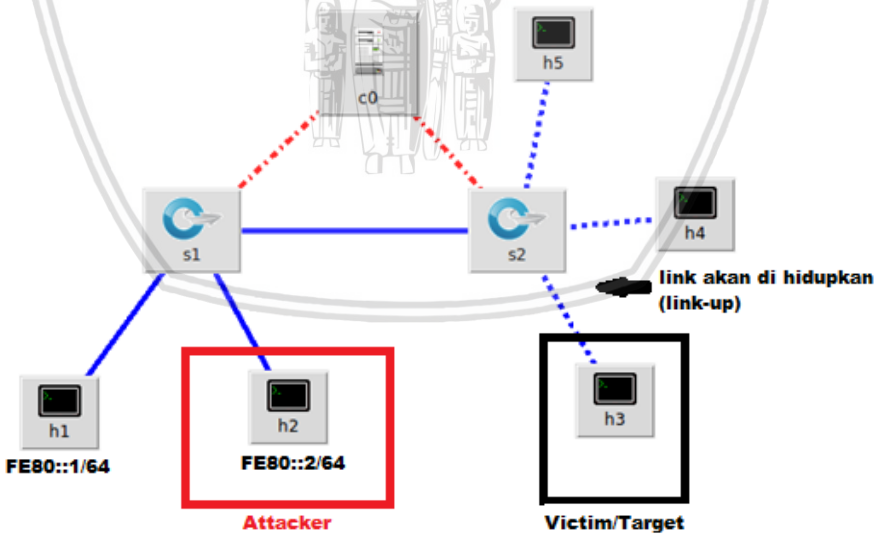
- Baris 1, merupakan inisiasi *method* "add_*flow*" yang membawa parameter *datapath*, *priority*, *match*, *action* serta waktu *idle timeout*.
- Baris 2-3, merupakan pembentukan obyek "ofproto" dengan nilai datapath.ofproto dari parameter bawaan, serta obyek "parser" dari yang merupakan hasil "parser" dari nilai "ofproto".
- Baris 4-5, merupakan pembentukan obyek "inst" yang nilai merupakan *action* yang ditentukan *main program* dalam penerusan paket.
- Baris 7-8, pembentukan obyek "mod" yang nilainya merupakan entri *flow* yang siap untuk dikirimkan ke *flow table*.
- Baris 9, mengirim variable "mod" ke method pembentukan *flow table*.

BAB 6
PENGUJIAN DAN ANALISIS

Pada bab ini akan menjelaskan hasil pengujian sistem yang telah di implementasikan beserta hasil pengujiannya. Pengujian dilaksanakan untuk menilai apakah kebutuhan serta fungsional yang telah ditetapkan telah tercukupi. Hasil dari pengujian dipergunakan untuk menilai apakah sistem berjalan sesuai dengan yang diinginkan serta dapat dijadikan sebagai acuan dalam penarikan kesimpulan penelitian ini.

6.1 Pengujian *Stateful Packet Inspection* Sebelum Pengembangan

Pengujian *Stateful Packet Inspection* ini dilakukan untuk menguji fungsional dari sistem sebelumnya dalam menangani paket IPv6, khususnya ICMPv6. *Stateful Packet Inspection* yang belum mendukung inpeksi terhadap ICMPv6 akan diuji dengan menjalankan skenario *duplicate address detection*, serangan *DAD DoS* serta percobaan ping menggunakan alamat IPv6. Pengujian ini dilakukan untuk memperoleh informasi apakah *Stateful Packet Inspection* yang belum dilakukan pengembangan mampu menangani proses *DAD* IPv6.



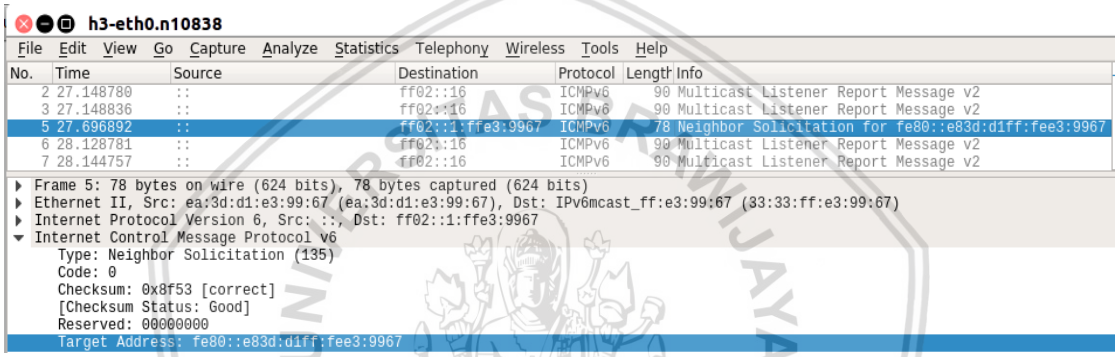
Gambar 6.1 Topologi Pengujian

Dalam pelaksanaannya, pengujian dilakukan menggunakan topologi yang telah dibuat pada bab perancangan yang terliihat pada gambar 6.1. *Link* yang statusnya aktif (*link-up*) adalah *link* pada *host* h1 dan h2, dimana *link* pada *host* lainnya (h3, h4

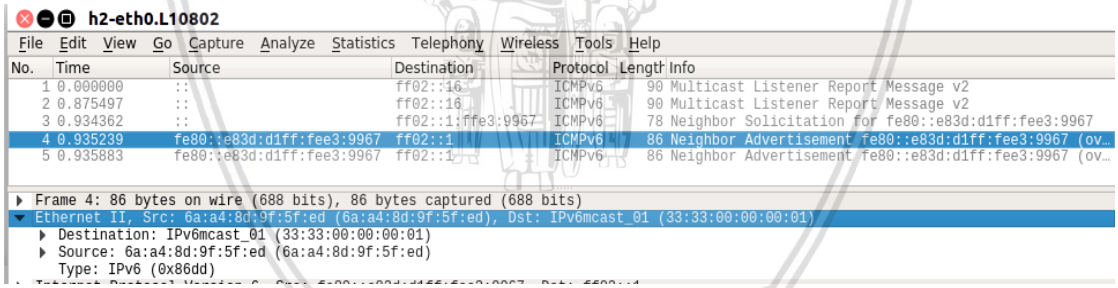
dan h5) diatur menjadi *down* (*link-down*) sehingga hanya *host* h1 dan h2 yang diawal telah aktif pada jaringan lokal.

6.1.1 Pengujian Serangan *Duplicate Address Detection DoS*

Untuk mengetahui apakah serangan *duplicate address detection DoS* dapat berajalan pada jaringan, maka dilakukan *capture* pada tiap *interface host* terkait (h1, h2 dan h3), dimana h3 berperan sebagai *host* baru yang ingin terhubung ke jaringan lokal, serta h2 menjalankan serangan *DAD DoS* pada terminal. Berikut merupakan hasil *capture* Wireshark dari *host* terhubung setelah dilakukan *link-up* terhadap *host* h3.

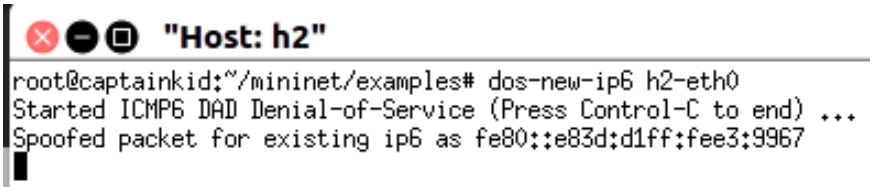


Gambar 6.2 Capture Wireshark h3



Gambar 6.3 Capture Wireshark h2

Gambar 6.2 menjelaskan bahwa *host* h3 mengirim *neighbor solicitation* dengan alamat *multicast* ke seluruh jaringan lokal yang terhubung. Pada gambar 6.3 terlihat bahwa *host* h2 menerima paket *neighbor solicitation* yang dikirim oleh h3 serta membalas paket tersebut dengan *spoofed neighbor advertisement*.



Gambar 6.4 Output Penyerang

```
Switch 1 has connected with OFP 1.3...
Deleting all Flow Table entries...
Rule will be constructed
Flow Table rules are sent to switch
Please set OFPMatch--> eth_type parameter in order to continue.
Flow Table rules are sent to switch
Eth_Type is set
LLDP rule will be added
datapath has joined
<module 'ryu.ofproto.ofproto_v1_3' from '/usr/local/lib/python2.7/dist-packages/ryu/ofproto/ofproto
_v1_3.pyc'>
<module 'ryu.ofproto.ofproto_v1_3_parser' from '/usr/local/lib/python2.7/dist-packages/ryu/ofproto/
ofproto_v1_3_parser.pyc'>
Switch 2 has connected with OFP 1.3...
Deleting all Flow Table entries...
Rule will be constructed
Flow Table rules are sent to switch
Please set OFPMatch--> eth_type parameter in order to continue.
Flow Table rules are sent to switch
Eth_Type is set
LLDP rule will be added
```

Gambar 6.5 Output kontroler

Gambar 6.5 merupakan output kontroler yang menunjukkan bahwa kontroler tidak mengenali proses *duplicate address detection* (*idle*) yang membuat penyerang mampu melakukan serangan pada proses *DAD* tanpa adanya pencegahan.

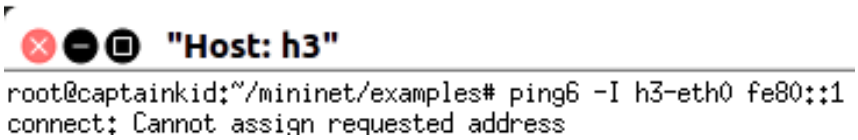
6.1.2 Pengujian Ping *Host* Baru

Pengujian *ping host* baru dilakukan untuk menguji fungsional dari sistem untuk mengetahui apakah alamat IPv6 yang digunakan oleh *host* baru (h3) setelah dilakukan simulasi serangan dapat digunakan pada jaringan lokal. Pengujian *ping* dilakukan dengan menjalankan *CLI host* h3 pada topologi Mininet yang telah ditetapkan sebelumnya. Berikut merupakan perintah yang dijalankan.

"$ ping6 -I h3-eth0 fe80::1"

Penjelasan:

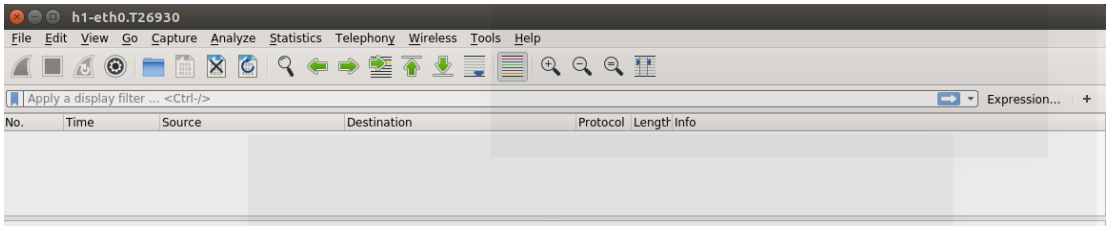
- "ping6", merupakan perintah dalam menjalankan *ping* dengan menggunakan protokol ICMPv6.
- "-I h3-eth0", "-I" merupakan parameter dari *network interface* yang digunakan oleh *host* h3.
- "fe80::1", merupakan alamat IPv6 dari *host* tujuan (h1)



Gambar 6.6 Pengujian ping

Gambar 6.6 menunjukkan bahwa *host* h3 tidak dapat melakukan ping terhadap "fe80::1" (*host* h1) yang alamatnya tidak terjangkau oleh h3. Hal ini terjadi karena paket *neighbor solicitation* dari h3 untuk menanyakan pemilik dari alamat

fe80::1 tidak di respon. Hal tersebut dapat ditunjukkan dari hasil *capture* Wireshark dari *host* h1.



Gambar 6.7 Capture Wireshark h1

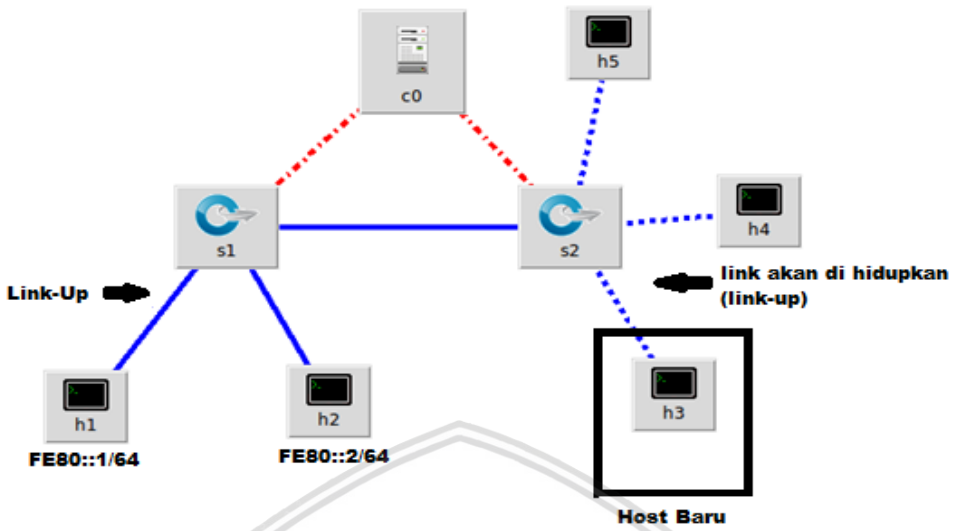
Gambar 6.7 menunjukkan terlihat hasil *capture* Wireshark dari *host* h1 yang merupakan alamat yang dituju, dimana *host* tersebut tidak menerima maupun mengirim paket (*neighbor advertisement* sebagai balasan *neighbor solicitation* dari h3 maupun paket *ping reply*).

6.2 Pengujian *Stateful Packet Inspection* Setelah Pengembangan

Pengujian ini dilakukan untuk menguji fungsionalitas dari pengembagan *Stateful Packet Inspection*. Tahap-tahap dalam pengujian sistem adalah proses *duplicate address detection* tanpa serangan, serangan *duplicate address detection DoS* (satu penyerang) dan serangan *duplicate address detection DoS* (banyak penyerang).

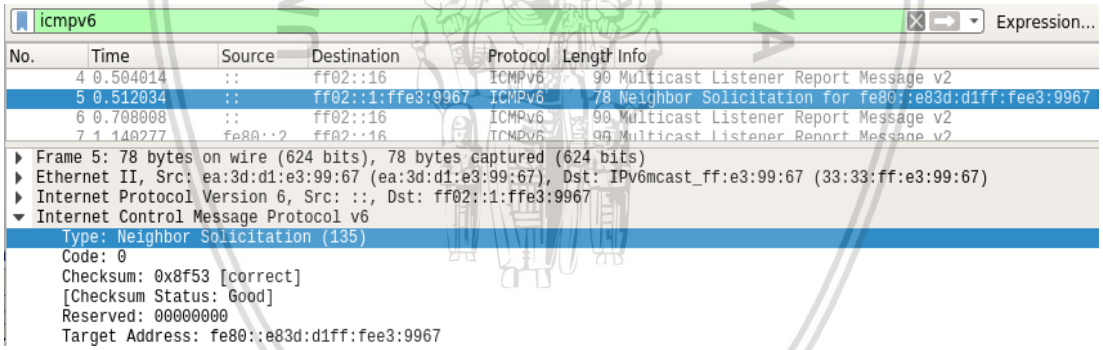
6.2.1 Pengujian Proses *Duplicate Address Detection* Tanpa Serangan

Pengujian ini dilakukan untuk menguji fungsional dari sistem yang diantaranya untuk mengetahui apakah sistem dapat medeteksi paket *neighbor solicitation* dan *neighbor advertisement* sesuai informasi yang dibawa (alamat IPv6 serta alamat MAC) serta sistem mampu melakukan pembaharuan *state table* sesuai dengan informasi yang dibawa oleh paket yang masuk. Dalam pelaksanaannya, pengujian dilakukan menggunakan topologi yang telah dibuat pada bab perancangan. *Link* yang statusnya aktif (*link-up*) adalah *link* pada *host* h1 dan h2, dimana *link* pada *host* lainnya (h3, h4 dan h5) diatur menjadi *down* (*link-down*) sehingga hanya *host* h1 dan h2 yang telah aktif pada jaringan lokal.

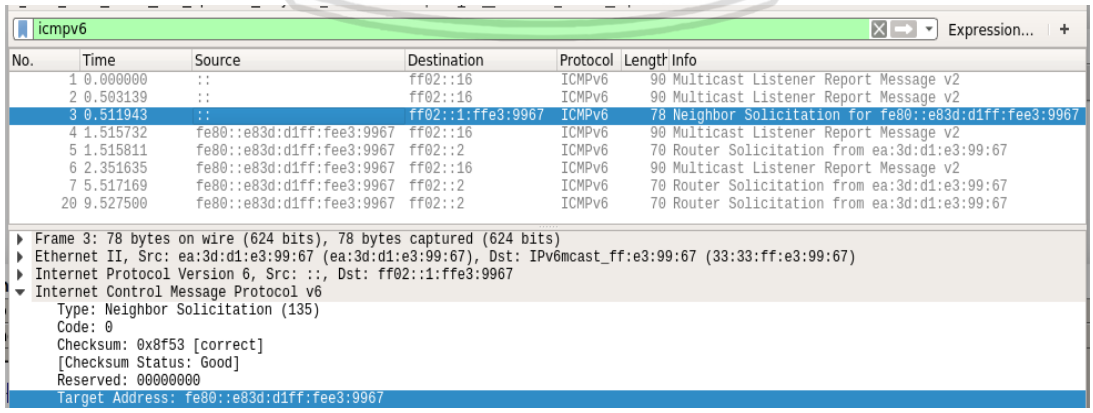


Gambar 6.8 Topologi Pengujian

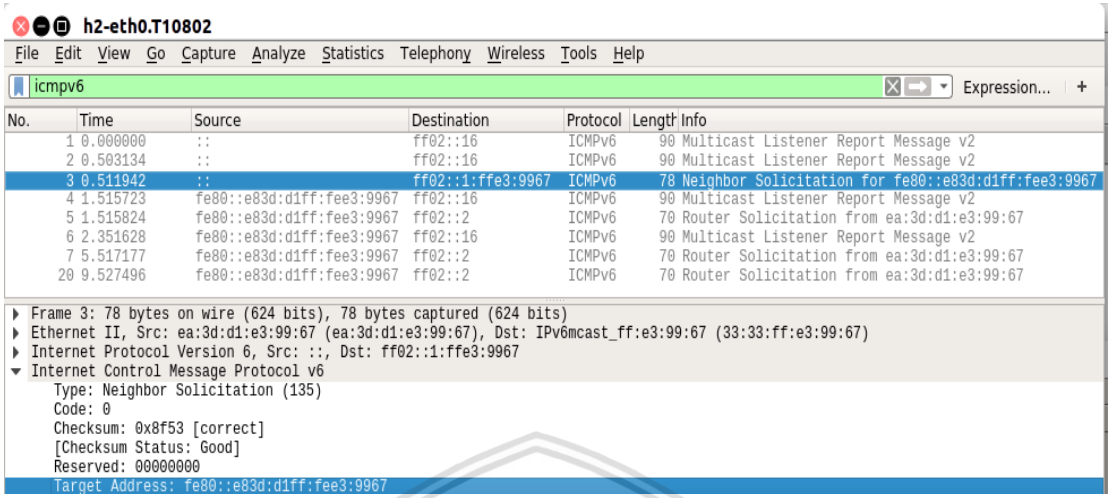
Untuk mengetahui proses *DAD* dapat berajalan pada jaringan, maka dilakukan *capture* pada tiap *interface host* (h1, h2 dan h3), dimana h3 berperan sebagai *host* baru yang ingin terhubung ke jaringan lokal. Berikut merupakan hasil *capture* Wireshark dari ketiga *host*.



Gambar 6.9 Capture h3-eth0

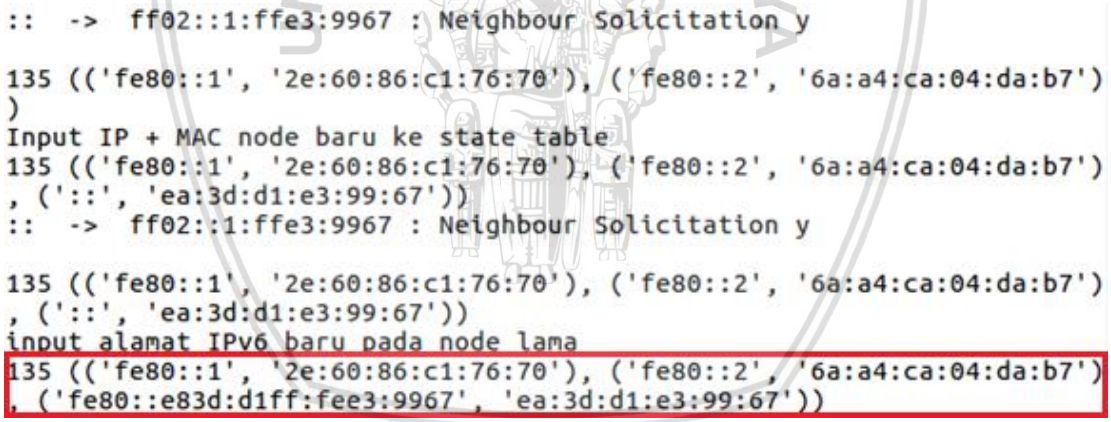


Gambar 6.10 Capture h1-eth0



Gambar 6.11 Capture h2-eth0

Gambar 6.9 menjelaskan bahwa *host* h3 melakukan *broadcast neighbor solicitation* ke seluruh *host* pada jaringan lokal dengan membawa informasi bila h3 ingin menggunakan alamat "fe80::e83d:d1ff:fee3:9967". Gambar 6.10 dan gambar 6.11 memperlihatkan hasil capture dari *host* h1 dan h2, dimana kedua *host* tersebut menerima *broadcast neighbor solicitation.*



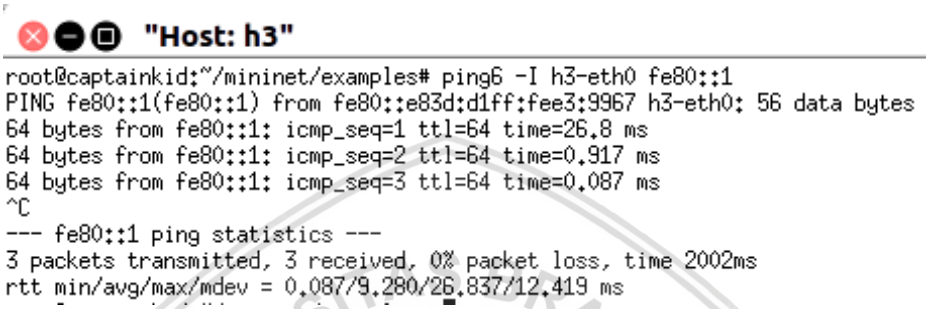
Gambar 6.12 *State table* pada controller

Berdasarkan hasil *filtering* sistem, maka sistem melakukan entri berupa informasi alamat IPv6 dan alamat MAC dari *host* baru ke dalam *state table.* Pengujian *ping host* baru dilakukan menguji fungsional dari sistem untuk mengetahui apakah alamat IPv6 yang digunakan oleh *host* baru (h3) setelah dilakukan proses *duplicate address detection* tanpa serangan dapat digunakan pada jaringan lokal. Pengujian *ping* dilakukan dengan menjalankan *CLI host* h3 pada topologi Mininet yang telah ditetapkan sebelumnya. Berikut merupakan perintah yang dijalankan.

"$ ping6 -I h3-eth0 fe80::1"

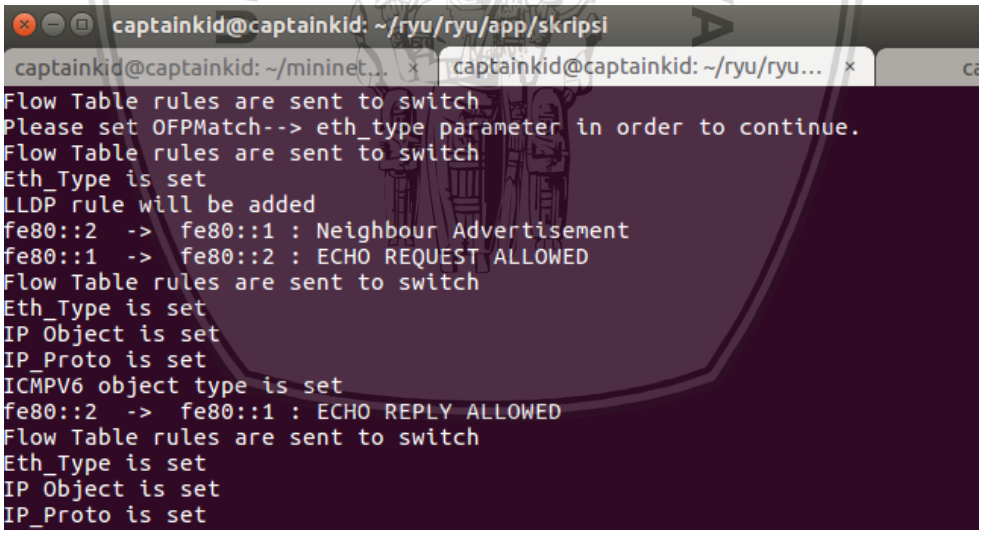
Penjelasan:

- “ping6”, merupakan perintah dalam menjalankan *ping* dengan menggunakan protokol ICMPv6.
- “-I h3-eth0”, “-I” merupakan parameter dari *network interface* yang digunakan oleh *host* h3.
- “fe80::1”, merupakan alamat IPv6 dari *host* tujuan (h1)



Gambar 6.13 Pengujian ping6 antar *host*

Berdasarkan gambar 6.13 dapat terlihat pengujian ping6 dari *host* asal (h1) menuju ke *host* tujuan (h2) mampu mengirimkan 11 paket *ping* dengan rasio *packet* loss sebesar 0%.



Gambar 6.14 Tampilan kontroler saat pengujian ping

Berdasarkan gambar 6.14 dapat terlihat kontroler dapat memahami paket ping6 dari *host* asal (h3) menuju ke *host* tujuan (h1) dan memberikan informasi bila entri *flow* dari paket tersebut telah ditambahkan kedalam *flow table*. Untuk mengetahui apakah entri *flow* yang terdapat pada *flow table* di *switch* tersimpan sesuai informasi yang dibawa paket, maka dilakukan pengecekan *flow table* dengan menjalankan perintah berikut.

“$ sudo ovs-ofctl dump-*flow*s -O OpenFlow13 s2”

Perintah tersebut akan menampilkan isi *flow table* yang menggunakan protokol OpenFlow 1.3 yang ada pada *switch* s2. Berikut adalah gambar yang menampilkan isi *flow table*.

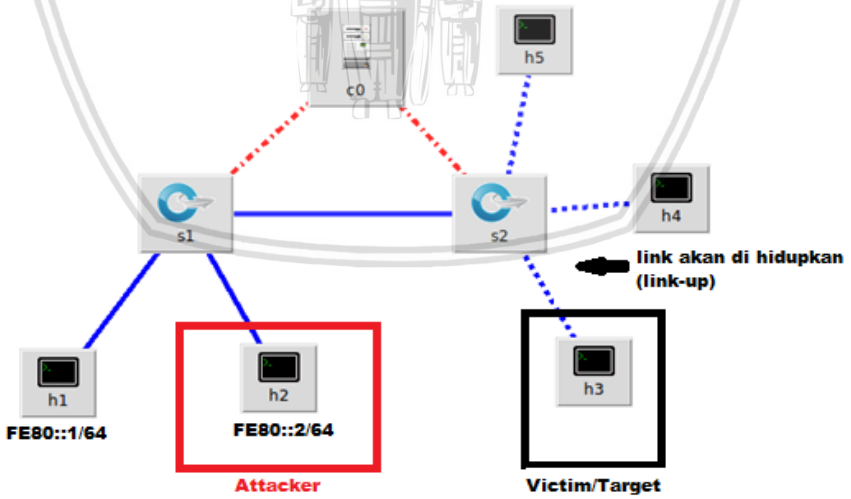
```
captainkid@captainkid:~$ sudo ovs-ofctl dump-flows -O OpenFlow13 s2
[sudo] password for captainkid:
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=515.308s, table=0, n_packets=2, n_bytes=236, idle_timeout=1800, priority=1001
,icmp6,in_port=1,ipv6_src=fe80::e83d:d1ff:fee3:9967,ipv6_dst=fe80::1,icmp_type=128 actions=output:2
 cookie=0x0, duration=515.296s, table=0, n_packets=2, n_bytes=236, idle_timeout=1800, priority=1002
,icmp6,in_port=2,ipv6_src=fe80::1,ipv6_dst=fe80::e83d:d1ff:fee3:9967,icmp_type=129 actions=output:1
 cookie=0x0, duration=2296.698s, table=0, n_packets=70, n_bytes=6658, idle_timeout=1800, priority=0
 actions=CONTROLLER:65535
```

Gambar 6.15 *Dump flow* dari *switch* s2

Berdasarkan gambar 6.15 dapat terlihat bahwa entri *flow* dari paket *ping* telah masuk berikut informasi detail yang dibawa paket tersebut (protokol, tipe ICMPv6 serta *IP* asal dan tujuan).

6.2.2 Pengujian *Duplicate Address Detection* DoS (Satu Penyerang)

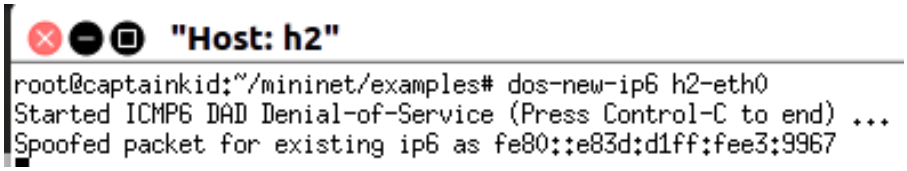
Pengujian ini dilakukan untuk menguji kinerja dari kemampuan *Stateful Packet Inspection* dalam menangani serangan *DAD DoS* dengan menggunakan satu *host* baru. Pada pengujian ini, sistem akan diuji melalui skema *spoofing* yang dilakukan *host* H2 dimana penyerang berpura-pura menggunakan alamat yang ingin digunakan *host* baru yang mencoba menggunakan dua alamat IPv6 yang berbeda. Pengujian ini dijalankan guna melihat apakah sistem mampu melakukan pencocokan informasi koneksi dari paket palsu terhadap *state table* yang sudah ada.



Gambar 6.16 Topologi Pengujian

Dalam pelaksanaannya, pengujian dilakukan menggunakan topologi yang telah dibuat pada bab perancangan. *Link* yang statusnya aktif (*link-up*) adalah *link* pada *host* h1 dan h2, dimana *link* pada *host* lainnya (h3, h4 dan h5) diatur menjadi *down* (*link-down*) sehingga hanya *host* h1 dan h2 yang diawal telah aktif pada jaringan lokal.

Simulasi serangan dilakukan dengan menjalankan tools THC-IPv6 toolkit pada *host* h2 dengan perintah “dos-new-ip6 h2-eth0”.



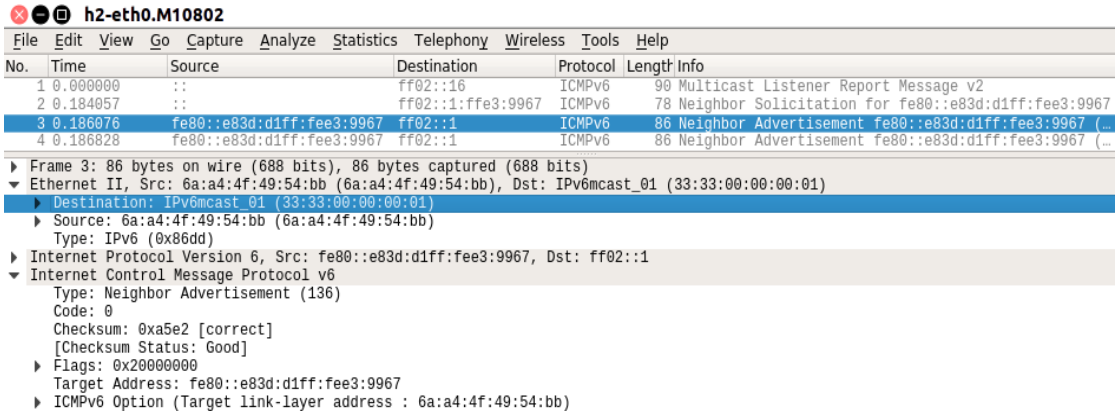
Gambar 6.17 *output tool* pada h2

Gambar 6.17 menunjukkan bahwa penyerang (h2) telah melakukan *spoofing* terhadap alamat IPv6 *host* h3 yang ingin terhubung ke dalam jaringan (*link-up*), yaitu “fe80::e83d:d1ff:fee3:9967”.

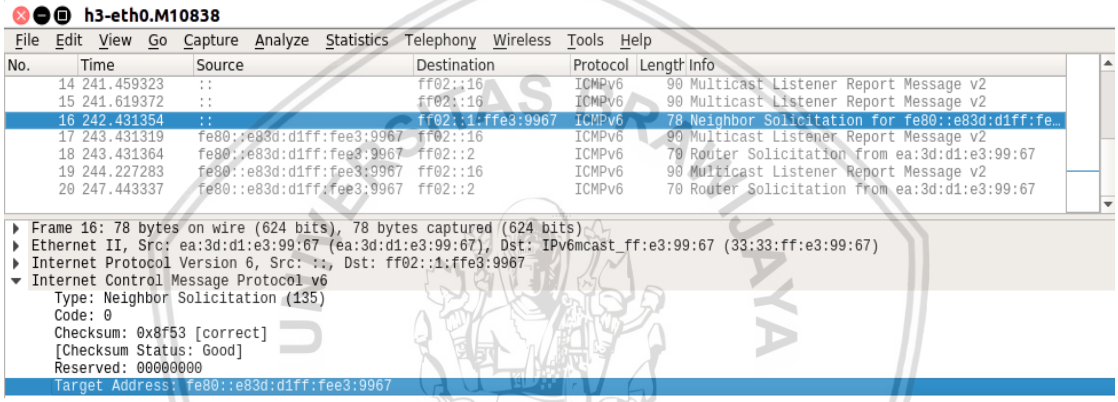
```
::  ->  ff02::1:ffe3:9967 : Neighbour Solicitation y
Input IP + MAC node baru ke state table
135 (('::', 'ea:3d:d1:e3:99:67'),)
::  ->  ff02::1:ffe3:9967 : Neighbour Solicitation y

135 (('::', 'ea:3d:d1:e3:99:67'),)
input alamat IPv6 baru pada node lama
135 (('fe80::e83d:d1ff:fee3:9967', 'ea:3d:d1:e3:99:67'),)
fe80::e83d:d1ff:fee3:9967  ->  ff02::1 : Neighbour Advertisement
Pengecekan Info dari Neighbor Advertisement
spoofed NA detection, drop NA packet 2

MAC Address  ->  6a:a4:4f:49:54:bb : entry ACL block list

fe80::e83d:d1ff:fee3:9967  ->  ff02::1 : Neighbour Advertisement
Pengecekan Info dari Neighbor Advertisement
spoofed NA detection, drop NA packet 2

MAC Address  ->  6a:a4:4f:49:54:bb : entry ACL block list
```

Gambar 6.18 *output controller* pada saat h3 terhubung

Pada gambar 6.18 terlihat bahwa terdapat 3 *state* koneksi yang tersimpan pada *state table* melalui hasil *filtering* oleh *Stateful Packet InspectionI* yang menandakan bahwa alamat IPv6 dari *host* 3 telah dikenali dan disimpan pada *state table*. Hal tersebut diperoleh dari inspeksi tiap koneksi yang terhubung. Berikut hasil capture Wireshark dari *host* baru (h3) dan *host* penyerang (h2).

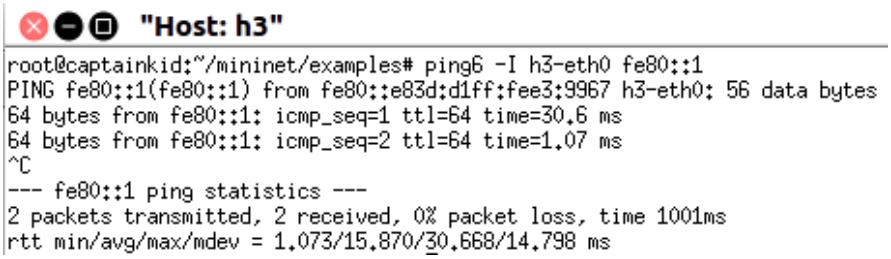


Gambar 6.19 capture Wireshark h2



Gambar 6.20 *capture* Wireshark h3

Gambar 6.19 memperlihatkan bahwa h2 melakukan *spoofing* dengan menyamar sebagai *host* dengan alamat "fe80::384d:d1ff:fee3:9967" dan mengirim *neighbor advertisement* sebagai balasan terhadap *broadcast neighbor solicitation* yang dilakukan *host* baru. Gambar 6.20 menunjukkan hasil capture h3, dimana *host* h3 tidak menerima paket *neighbor advertisement* yang dikirim h2. Hasil uji sistem ini dapat dipastikan dengan melakukan *ping* dari h3 ke h1 untuk menunjukkan apakah h3 sudah dapat terhubung dan berkomunikasi pada jaringan lokal.



Gambar 6.21 *ping* h3 ke h1

Gambar 6.21 menjelaskan bahwa *host* h3 dapat menggunakan alamat IPv6 yang ingin digunakan dengan menjalankan perintah ping dari *host* h3 menuju *host*

h1). Sistem juga mampu menambahkan entri *ACL* (*Access Control List*) dari informasi koneksi yang terdeteksi sebagai suatu serangan yang disimpan didalam library *ACL*.

```
(('6a:a4:4f:49:54:bb', '-', '-', 'ICMP', 'BLOCK'),)
```

Gambar 6.22 *ACL Rules*

Gambar 6.22 menjelaskan bahwa kontroler dapat melakukan input entri ACL terhadap koneksi yang terdeteksi sebagai serangan dengan action 'BLOCK', dimana MAC *address* yang digunakan penyerang 6a:a4:4f:49:54:bb) telah terdaftar di daftar *ACL Rules*. Selanjutnya dilakukan perubahan pada alamat IPv6 yang digunakan oleh h3, pengujian ini dilakukan guna menguji fungsionalitas dari sistem, apakah sistem mampu melakukan update *state table* bila terdapat *host* yang ingin memperbaharui alamat IPv6 yang digunakan. Untuk menjalankan proses tersebut, dilakukan perubahan pada *network interface* pada h3, dengan menjalankan perintah pada *CLI* Mininet sebagai berikut.

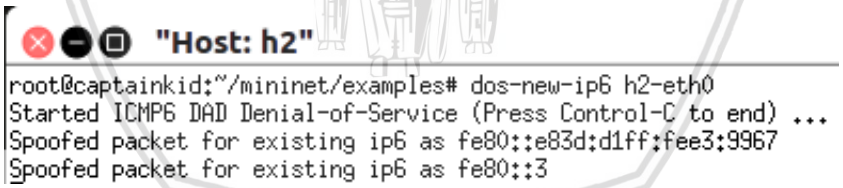
"h3 iconfig h3-eth0 inet6 del "alamat IPv6 lama" "

"h3 iconfig h3-eth0 inet6 add "alamat IPv6 baru""

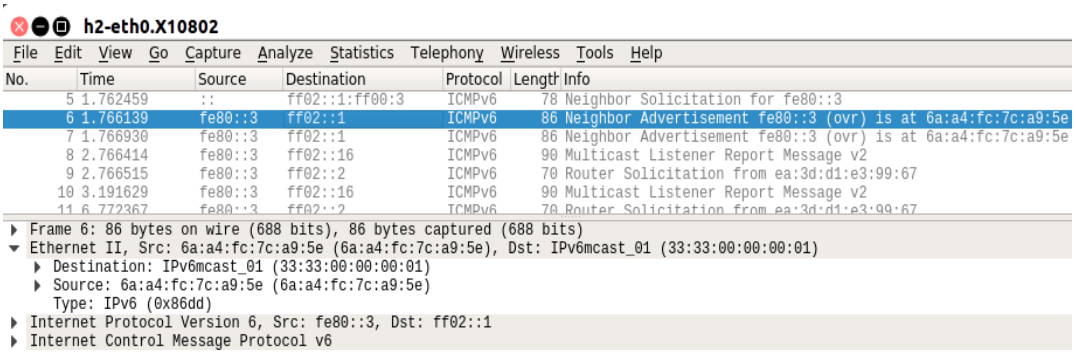
```
mininet> h3 ifconfig h3-eth0 inet6 del fe80::e83d:d1ff:fee3:9967/64
mininet> h3 ifconfig h3-eth0 inet6 add fe80::3/64
```

Gambar 6.23 proses penambahan dan penghapusan alamat IPv6

Setelah melakukan konfigurasi perubahan alamat IPv6 pada *host* h3, maka penyerang (*host* h2) secara otomatis melakukan *spoofing neighbor advertisement* terhadap alamat IPv6 baru yang diminta oleh *host* h3 seperti yang ditampilkan pada gambar 6.24.

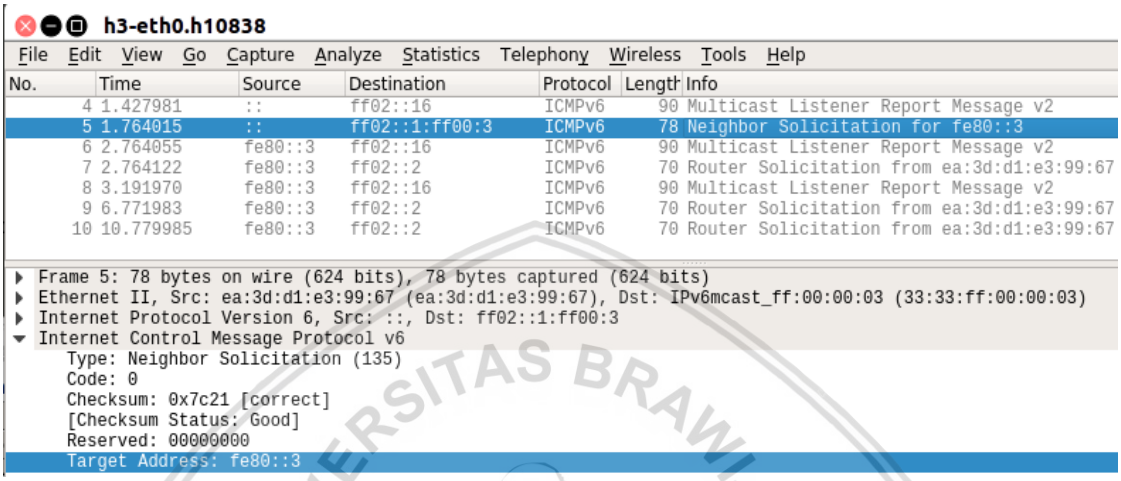


Gambar 6.24 tools pada h2

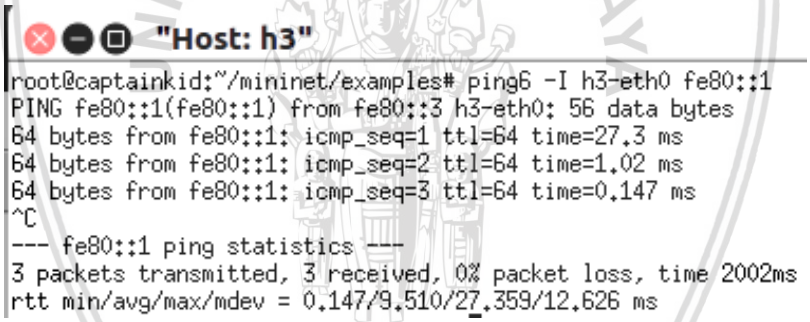


Gambar 6.25 *capture* Wireshark pada h2

Pada gambar 6.25 *host* h2 yang merupakan penyerang melakukan *spoofing* yang berpura-pura menjadi *host* yang memilki alamat “fe80::3”. Gambar 6.21 juga menunjukkan hasil *capture* dari *network interface* h2, dimana h2 menerima *neighbor* solicitation dari h3 yang ingin menggunakan alamat IPv6 “fe80::3” serta membalasnya dengan *spoofed neighbor advertisement.*



Gambar 6.26 hasil *capture* Wireshark h3



Gambar 6.27 hasil *ping* menggunakan alamat baru

Gambar 6.27 menunjukkan hasil *capture* Wireshark dari *host* h3, dimana *host* h3 tidak menerima paket *neighbor advertisement* dari *host* penyerang. *host* h3 tetap bisa menggunakan alamat IPv6 “fe80::3”, hal itu ditunjukkan dari hasil *ping host* h3 menggunakan alamat “fe80::3” ke h1 yang ditunjukkan pada gambar 6.23.

```
6a:a4:fc:7c:a9:5e (('6a:a4:fc:7c:a9:5e', '-', '-', 'ICMP', 'BLOCK'),)
6a:a4:4f:49:54:bb (('6a:a4:4f:49:54:bb', '-', '-', 'ICMP', 'BLOCK'),)
```

Gambar 6.28 *ACL List*

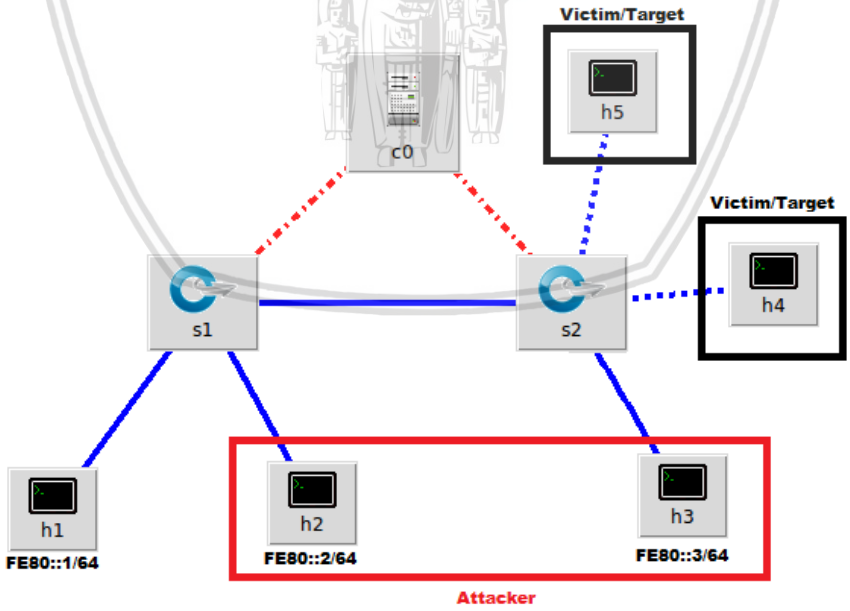
Gambar 6.28 menjelaskan bahwa kontroler dapat melakukan input entri ACL terhadap koneksi yang terdeteksi sebagai serangan dengan action ‘BLOCK’, dimana MAC *address* yang digunakan penyerang (6a:a4:fc:7c:a9:5e) telah terdaftar di daftar *ACL Rules*.

```
captainkid@captainkid:~$ sudo ovs-ofctl dump-flows -O OpenFlow13 s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=1692.827s, table=0, n_packets=0, n_bytes=0, idle_timeout=1800, priority=10000
,dl_type=0x88cc actions=output:32
 cookie=0x0, duration=149.831s, table=0, n_packets=2, n_bytes=236, idle_timeout=1800, priority=1001
,icmp6,in_port=3,ipv6_src=fe80::e881:3bff:fe77:3b6f,ipv6_dst=fe80::1,icmp_type=128 actions=output:2
 cookie=0x0, duration=149.827s, table=0, n_packets=2, n_bytes=236, idle_timeout=1800, priority=1002
,icmp6,in_port=2,ipv6_src=fe80::1,ipv6_dst=fe80::e881:3bff:fe77:3b6f,icmp_type=129 actions=output:3
 cookie=0x0, duration=111.497s, table=0, n_packets=2, n_bytes=236, idle_timeout=1800, priority=1001
,icmp6,in_port=3,ipv6_src=fe80::2416:43ff:febc:f713,ipv6_dst=fe80::1,icmp_type=128 actions=output:2
 cookie=0x0, duration=111.493s, table=0, n_packets=2, n_bytes=236, idle_timeout=1800, priority=1002
,icmp6,in_port=2,ipv6_src=fe80::1,ipv6_dst=fe80::2416:43ff:febc:f713,icmp_type=129 actions=output:3
 cookie=0x0, duration=1692.827s, table=0, n_packets=39, n_bytes=3576, idle_timeout=1800, priority=0
 actions=CONTROLLER:65535

captainkid@captainkid:~$ sudo ovs-ofctl dump-flows -O OpenFlow13 s2
[sudo] password for captainkid:
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=1347.860s, table=0, n_packets=0, n_bytes=0, idle_timeout=1800, priority=10000
,dl_type=0x88cc actions=output:32
 cookie=0x0, duration=670.444s, table=0, n_packets=1, n_bytes=118, idle_timeout=1800, priority=1001
,icmp6,in_port=1,ipv6_src=fe80::e83d:d1ff:fee3:9967,ipv6_dst=fe80::1,icmp_type=128 actions=output:2
 cookie=0x0, duration=670.430s, table=0, n_packets=1, n_bytes=118, idle_timeout=1800, priority=1002
,icmp6,in_port=2,ipv6_src=fe80::1,ipv6_dst=fe80::e83d:d1ff:fee3:9967,icmp_type=129 actions=output:1
 cookie=0x0, duration=61.276s, table=0, n_packets=2, n_bytes=236, idle_timeout=1800, priority=1001,
icmp6,in_port=1,ipv6_src=fe80::3,ipv6_dst=fe80::1,icmp_type=128 actions=output:2
 cookie=0x0, duration=61.255s, table=0, n_packets=2, n_bytes=236, idle_timeout=1800, priority=1002,
icmp6,in_port=2,ipv6_src=fe80::1,ipv6_dst=fe80::3,icmp_type=129 actions=output:1
 cookie=0x0, duration=1347.860s, table=0, n_packets=45, n_bytes=4003, idle_timeout=1800, priority=0
 actions=CONTROLLER:65535
```

Gambar 6.29 *Flow rules* pada *switch*

Gambar 6.29 menunjukkan sistem mampu membentuk *flow rules* dari proses komunikasi antar *host* (ping), dimana match rules dari flow yang dibentuk didasarkan dari hasil informasi koneksi yang telah diinspeksi sebelumnya.

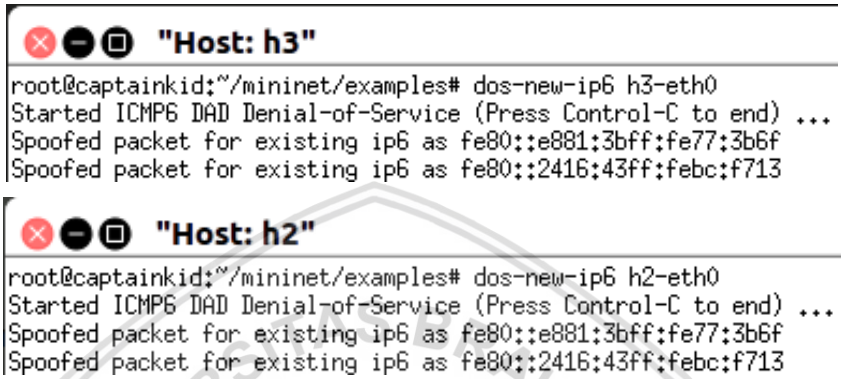
6.2.3 Pengujian *Duplicate Address Detection* DoS (Dua Penyerang)



Gambar 6.30 Topologi Pengujian

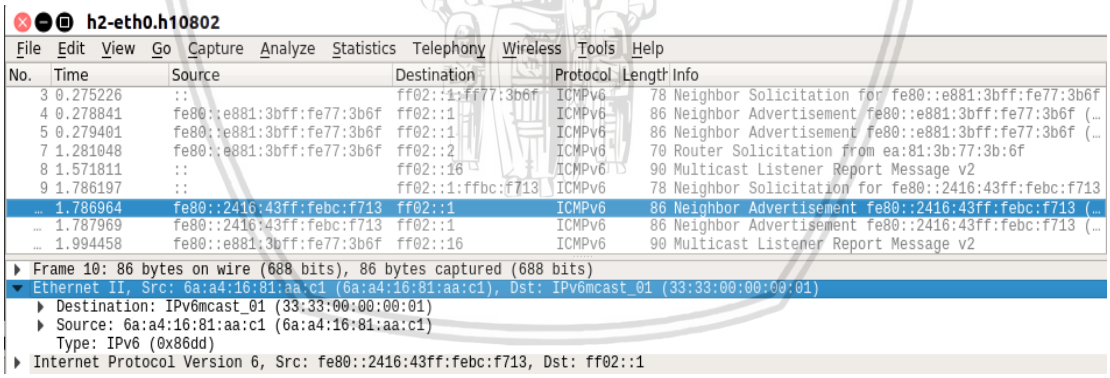
Pengujian ini dilakukan untuk menguji kinerja dari kemampuan *Stateful Packet Inspection* dalam menangani serangan *DAD DoS* pada pembentukan lebih dari satu

host baru. Pada pengujian ini, sistem akan diuji melalui skema *spoofing* yang dilakukan *host* h2 dan h3 yang terlihat pada gambar 6.30, dimana penyerang berpura-pura menggunakan alamat yang ingin digunakan *host* baru yang mencoba menggunakan alamat IPv6 yang berbeda. Pengujian ini dijalankan guna melihat apakah sistem mampu melakukan pencocokan informasi koneksi dari paket palsu terhadap *state table* yang sudah ada.

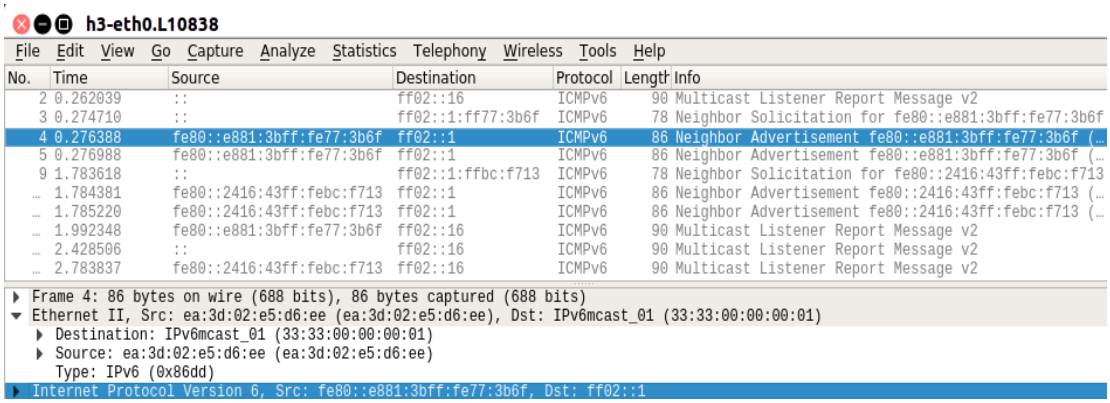


Gambar 6.31 *output* penyerang (h2 dan h3)

Gambar 6.31 menunjukkan bahwa *host* h2 dan h3 berhasil mendeteksi adanya *host* baru yang masuk serta melakukan *spoofing* pada alamat IPv6 yang ingin digunakan oleh *host* baru. Proses *spoofing* juga ditunjukkan pada hasil *capture* Wireshark terhadap *host* penyerang.

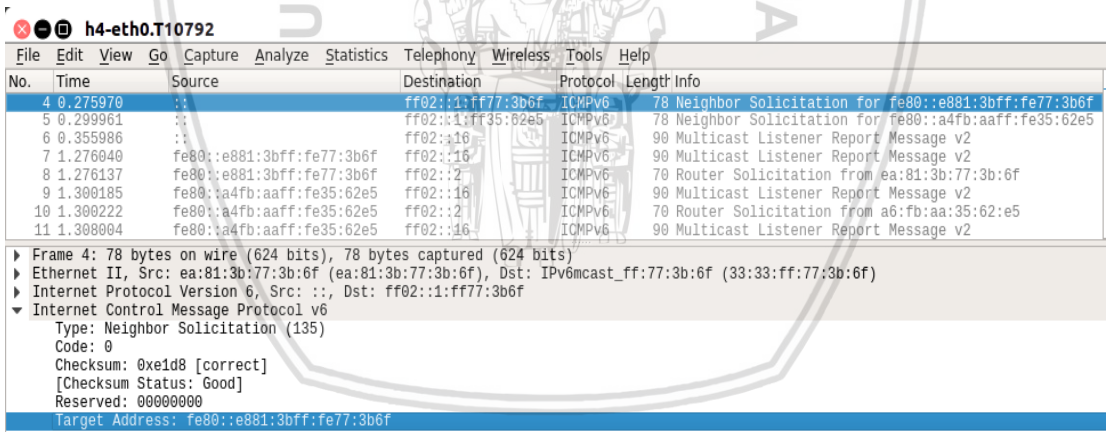


Gambar 6.32 *capture* Wireshark h2

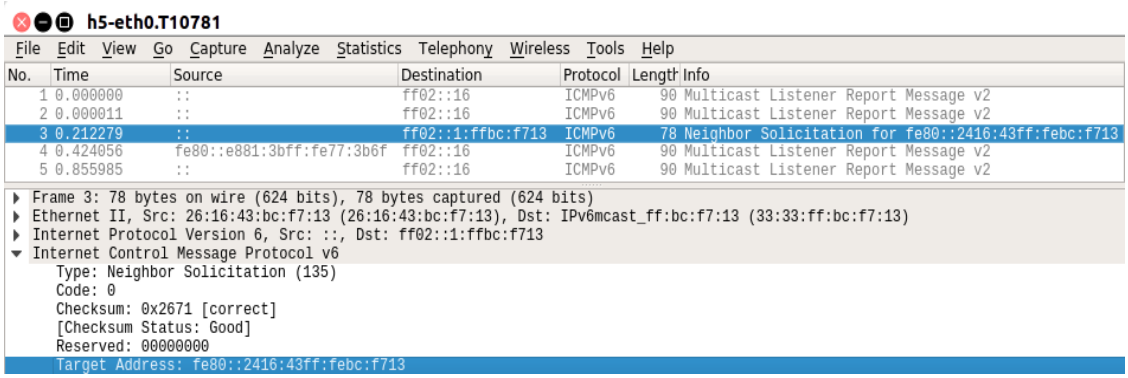


Gambar 6.33 *capture* Wireshark h3

Gambar 6.32 dan gambar 6.33 menunjukkan bahwa h2 dan h3 menerima *broadcast* paket *neighbor solicitation* dari kedua *host* baru (h4 dan h5) yang ingin menggunakan alamat IPv6 baru, yaitu "fe80::e881:3bff:fe77:3b67" serta "fe80::2416:43ff:febc:f713". Kedua gambar diatas juga menunjukkan bahwa h2 dan h3 selaku penyerang berhasil melakukan *spoofing neighbor advertisement* yang dikirim sebagai balasan dari *neighbor solicitation* sebelumnya. Namun paket *neighbor advertisement* dari kedua *host* penyerang tidak di terima oleh kedua *host* baru, hal itu dapat ditunjukkan pada hasil *capture host* h3 dan h5 pada kedua gambar dibawah.



Gambar 6.34 *capture* Wireshark h4



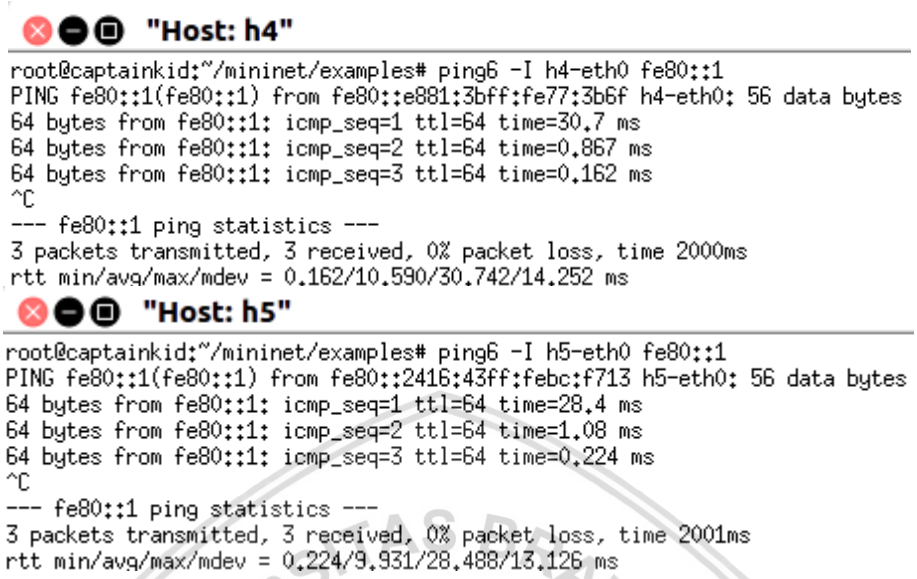
Gambar 6.35 *capture* Wireshark h5

Gambar 6.34 dan 6.35 menunjukkan bahwa kedua *host* baru tidak menerima paket *neighbor advertisement* yang telah dikirim oleh *host* penyerang (h2 dan h3). Berikut merupakan tampilan keluaran dari kontroler.

```
::  ->  ff02::1:ffbc:f713 : Neighbour Solicitation y

135 (('fe80::e881:3bff:fe77:3b6f', 'ea:81:3b:77:3b:6f'),)
Input IP + MAC node baru ke state table
135 (('fe80::e881:3bff:fe77:3b6f', 'ea:81:3b:77:3b:6f'), ('::', '26:16:43:bc:f7:13'))
::  ->  ff02::1:ffbc:f713 : Neighbour Solicitation y

135 (('fe80::e881:3bff:fe77:3b6f', 'ea:81:3b:77:3b:6f'), ('::', '26:16:43:bc:f7:13'))
input alamat IPv6 baru pada node lama
135 (('fe80::e881:3bff:fe77:3b6f', 'ea:81:3b:77:3b:6f'), ('fe80::2416:43ff:febc:f713', '26:16:43:bc
:f7:13'))
fe80::2416:43ff:febc:f713  ->  ff02::1 : Neighbour Advertisement
Pengecekan Info dari Neighbor Advertisement
spoofed NA detection, drop NA packet 2

MAC Address  ->  ea:3d:4d:d7:e1:2e : entry ACL block list

fe80::2416:43ff:febc:f713  ->  ff02::1 : Neighbour Advertisement
Pengecekan Info dari Neighbor Advertisement
spoofed NA detection, drop NA packet 2
```

Gambar 6.36 *output controller* pada saat h4 dan h5 masuk

Dari gambar 6.36 terlihat bahwa kontroler mampu melakukan *filtering* dengan memberikan notifikasi *drop spoofed packet* pada saat medeteksi adanya serangan yang dilakukan *host* h2 dan h3. Untuk mengetahui bahwa proses *DAD DoS* berhasil dicegah, maka dilakukan proses ping dari kedua *host* baru menuju ke *host* lama (h1).



Gambar 6.37 Ping *host* baru menuju h1

Gambar 6.37 memperlihatkan bahwa proses *ping* dari *host* h4 ke *host* h1 dan *ping* dari *host* h5 ke *host* h1 dapat dilakukan, dimana kedua *host* baru dapat menggunakan alamat IPv6 yang mereka bawa pada proses *DAD* sebelumnya. Dengan berhasilnya proses *ping* yang dilakukan kedua *host* baru, maka pembentukan *flow rules* pada *switch* juga dijalankan.

```
captainkid@captainkid:~$ sudo ovs-ofctl dump-flows -O OpenFlow13 s2
[sudo] password for captainkid:
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=1654.570s, table=0, n_packets=0, n_bytes=0, idle_timeout=1800, priority=10000
,dl_type=0x88cc actions=output:32
 cookie=0x0, duration=111.582s, table=0, n_packets=2, n_bytes=236, idle_timeout=1800, priority=1001
,icmp6,in_port=4,ipv6_src=fe80::e881:3bff:fe77:3b6f,ipv6_dst=fe80::1,icmp_type=128 actions=output:2
 cookie=0x0, duration=111.569s, table=0, n_packets=2, n_bytes=236, idle_timeout=1800, priority=1002
,icmp6,in_port=2,ipv6_src=fe80::1,ipv6_dst=fe80::e881:3bff:fe77:3b6f,icmp_type=129 actions=output:4
 cookie=0x0, duration=73.248s, table=0, n_packets=2, n_bytes=236, idle_timeout=1800, priority=1001,
icmp6,in_port=3,ipv6_src=fe80::2416:43ff:febc:f713,ipv6_dst=fe80::1,icmp_type=128 actions=output:2
 cookie=0x0, duration=73.235s, table=0, n_packets=2, n_bytes=236, idle_timeout=1800, priority=1002,
icmp6,in_port=2,ipv6_src=fe80::1,ipv6_dst=fe80::2416:43ff:febc:f713,icmp_type=129 actions=output:3
 cookie=0x0, duration=1654.570s, table=0, n_packets=39, n_bytes=3576, idle_timeout=1800, priority=0
 actions=CONTROLLER:65535_

captainkid@captainkid:~$ sudo ovs-ofctl dump-flows -O OpenFlow13 s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=1692.827s, table=0, n_packets=0, n_bytes=0, idle_timeout=1800, priority=10000
,dl_type=0x88cc actions=output:32
 cookie=0x0, duration=149.831s, table=0, n_packets=2, n_bytes=236, idle_timeout=1800, priority=1001
,icmp6,in_port=3,ipv6_src=fe80::e881:3bff:fe77:3b6f,ipv6_dst=fe80::1,icmp_type=128 actions=output:2
 cookie=0x0, duration=149.827s, table=0, n_packets=2, n_bytes=236, idle_timeout=1800, priority=1002
,icmp6,in_port=2,ipv6_src=fe80::1,ipv6_dst=fe80::e881:3bff:fe77:3b6f,icmp_type=129 actions=output:3
 cookie=0x0, duration=111.497s, table=0, n_packets=2, n_bytes=236, idle_timeout=1800, priority=1001
,icmp6,in_port=3,ipv6_src=fe80::2416:43ff:febc:f713,ipv6_dst=fe80::1,icmp_type=128 actions=output:2
 cookie=0x0, duration=111.493s, table=0, n_packets=2, n_bytes=236, idle_timeout=1800, priority=1002
,icmp6,in_port=2,ipv6_src=fe80::1,ipv6_dst=fe80::2416:43ff:febc:f713,icmp_type=129 actions=output:3
 cookie=0x0, duration=1692.827s, table=0, n_packets=39, n_bytes=3576, idle_timeout=1800, priority=0
 actions=CONTROLLER:65535
```

Gambar 6.38 *Flow rules* pada *switch*

Gambar 6.38 menunjukkan melalui perintah “ovs-ofctl dump-flows” pada *switch* “s2” dan “s1” bahwa *flow* dari koneksi *host* baru telah dapat terbentuk dimana *match* dari

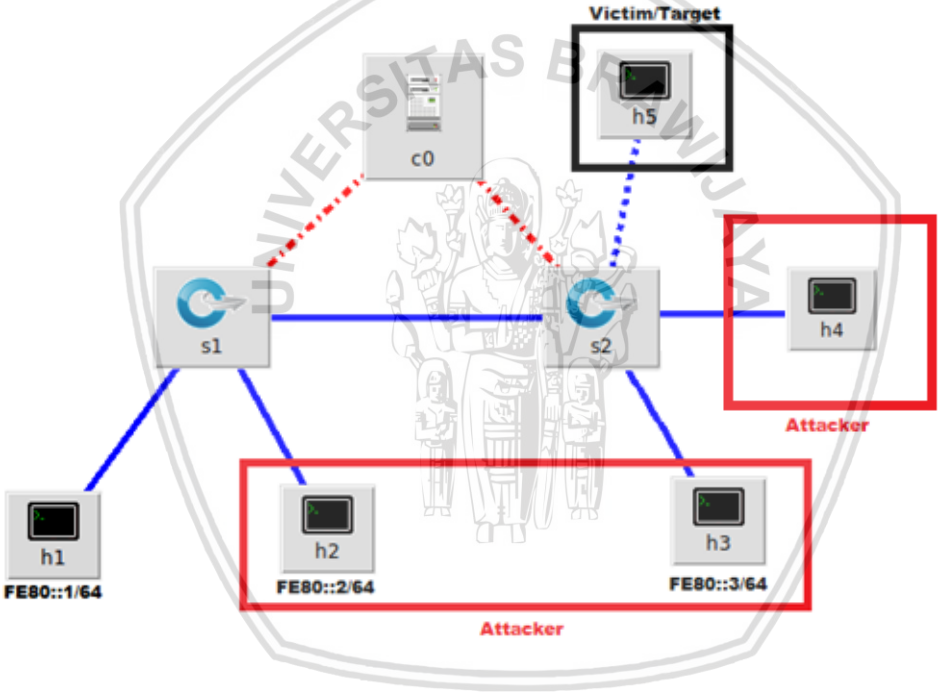
flow tersebut (IPv6 *destination*, IPv6 *source*, tipe ICMPv6 serta *action* yang dijalankan) disesuaikan oleh hasil *filtering* IPv6.

```
(('ea:3d:4d:d7:e1:2e', '-', '-', 'ICMP', 'BLOCK'),)
(('ea:3d:02:e5:d6:ee', '-', '-', 'ICMP', 'BLOCK'),)
(('6a:a4:16:81:aa:c1', '-', '-', 'ICMP', 'BLOCK'),)
(('6a:a4:04:76:ef:6d', '-', '-', 'ICMP', 'BLOCK'),)
```

Gambar 6.39 *ACL* rules pada kontroler

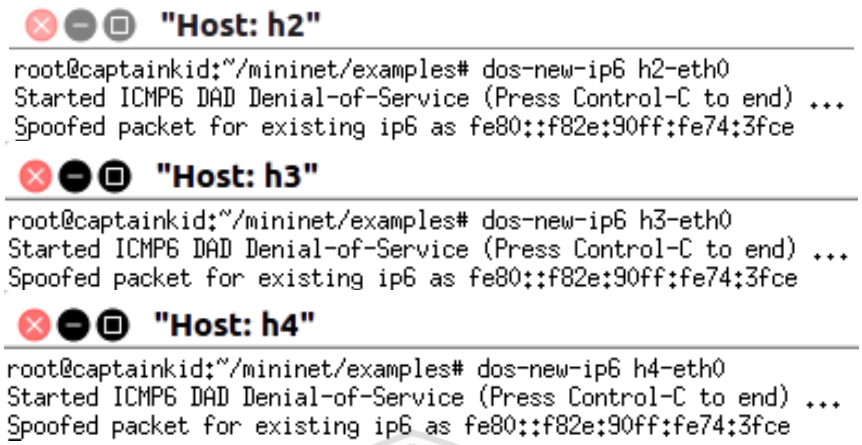
Gambar 6.39 menunjukkan bahwa kontroler dapat melakukan input entri ACL terhadap koneksi yang terdeteksi sebagai serangan dengan action 'BLOCK', dimana MAC *address* yang digunakan penyerang telah terdaftar di daftar *ACL Rules*.

6.2.4 Pengujian *Duplicate Address Detection* DoS (Tiga Penyerang)



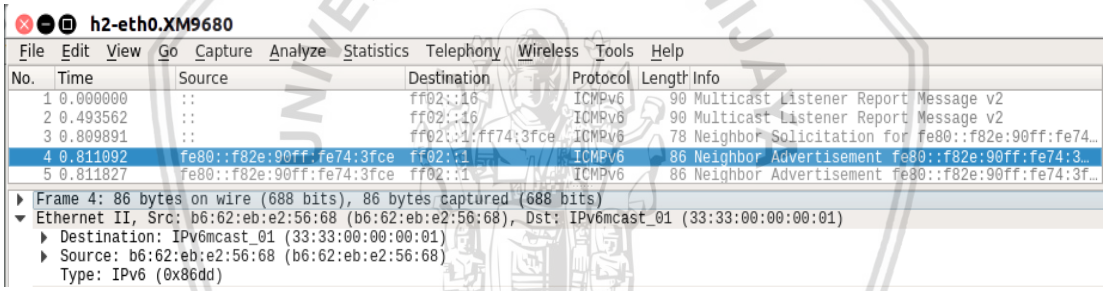
Gambar 6.40 Topologi Pengujian

Pengujian ini dilakukan untuk menguji kinerja dari kemampuan *Stateful Packet Inspection* dalam menangani serangan *DAD DoS* pada pembentukan lebih dari satu *host* baru. Pada pengujian ini, sistem akan diuji melalui skema *spoofing* yang dilakukan *host* h2, h3 dan h4, dimana penyerang berpura-pura menggunakan alamat yang ingin digunakan *host* baru yang mencoba menggunakan alamat IPv6 yang berbeda. Pengujian ini dijalankan guna melihat apakah sistem mampu melakukan pencocokan informasi koneksi dari paket palsu terhadap *state table* yang sudah ada.

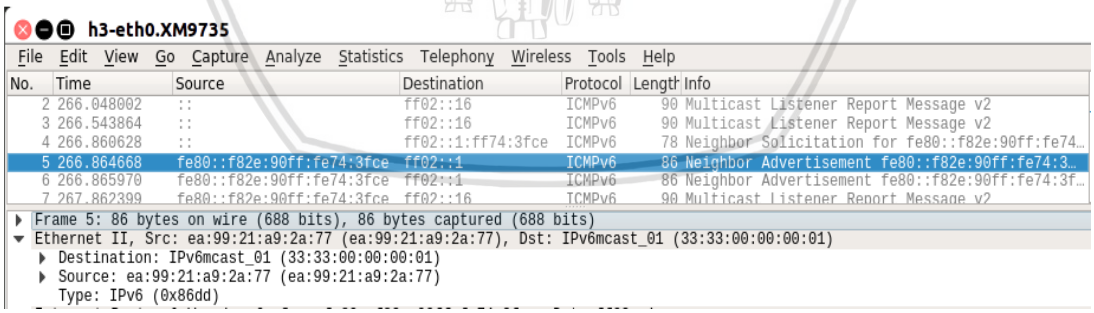


Gambar 6.41 *output* penyerang (h2, h3 dan h4)

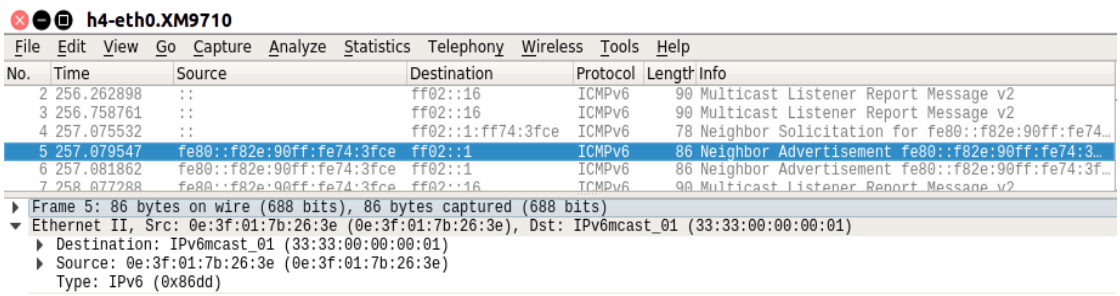
Gambar 6.41 menunjukkan bahwa *host* h2, h3 dan h4 berhasil mendeteksi adanya *host* baru yang masuk serta melakukan *spoofing* pada alamat IPv6 yang ingin digunakan oleh *host* baru. Proses *spoofing* juga ditunjukkan pada hasil *capture* Wireshark terhadap *host* penyerang.



Gambar 6.42 *capture* Wireshark h2

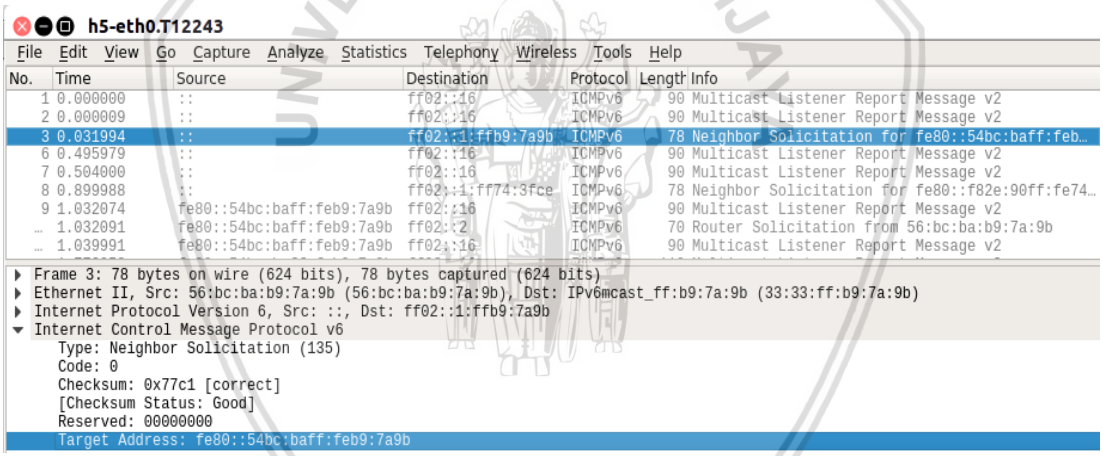


Gambar 6.43 *capture* Wireshark h3



Gambar 6.44 *capture* Wireshark h4

Gambar 6.42, gambar 6.43 dan gambar 6.44 menunjukkan bahwa h2, h3 dan h4 menerima *broadcast* paket *neighbor solicitation* dari kedua *host* baru (*host* h5) yang ingin menggunakan alamat IPv6 baru, yaitu "fe80::f82e:90ff:fe74:37ce". Ketiga gambar diatas juga menunjukkan bahwa h2, h3 dan h4 selaku penyerang berhasil melakukan *spoofing neighbor advertisement* yang dikirim sebagai balasan dari *neighbor solicitation* sebelumnya. Namun paket *neighbor advertisement* dari kedua *host* penyerang tidak di terima oleh *host* baru, hal itu dapat ditunjukkan pada hasil *capture host* h3 dan h5 pada kedua gambar dibawah.



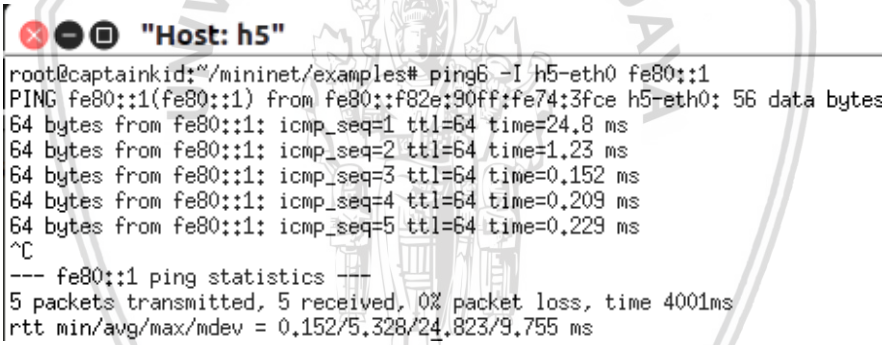
Gambar 6.45 *capture* Wireshark h5

Gambar 6.45 menunjukkan bahwa *host* baru tidak menerima paket *neighbor advertisement* yang telah dikirim oleh *host* penyerang (h2, h3 dan h4). Berikut merupakan tampilan keluaran dari kontroler.

```
fe80::f82e:90ff:fe74:3fce  ->  ff02::1 : Neighbour Advertisement
Pengecekan Info dari Neighbor Advertisement
spoofed NA detection, drop NA packet 2

MAC Address  ->  b6:62:eb:e2:56:68 : entry ACL block list
fe80::f82e:90ff:fe74:3fce  ->  ff02::1 : Neighbour Advertisement
Pengecekan Info dari Neighbor Advertisement
spoofed NA detection, drop NA packet 2

MAC Address  ->  0e:3f:01:7b:26:3e : entry ACL block list
fe80::f82e:90ff:fe74:3fce  ->  ff02::1 : Neighbour Advertisement
Pengecekan Info dari Neighbor Advertisement
spoofed NA detection, drop NA packet 2

MAC Address  ->  ea:99:21:a9:2a:77 : entry ACL block list
```

Gambar 6.46 *output controller* pada saat *host* h5 masuk

Dari gambar 6.46 terlihat bahwa kontroler melakukan *filtering* dengan memberikan notifikasi *drop spoofed packet* pada saat medeteksi adanya serangan yang dilakukan *host* h2, h3 dan h4. Untuk mengetahui bahwa proses *DAD DoS* berhasil dicegah, maka dilakukan proses ping dari kedua *host* baru menuju ke *host* lama (h1).



Gambar 6.47 Ping *host* baru menuju h1

Gambar 6.47 memperlihatkan bahwa proses *ping* dari *host* h5 ke *host* h1 dapat dilakukan, dimana kedua *host* baru dapat menggunakan alamat IPv6 yang mereka bawa pada proses *DAD* sebelumnya. Dengan berhasilnya proses *ping* yang dilakukan kedua *host* baru, maka pembentukan *flow rules* pada *switch* juga dijalankan.

```
captainkid@captainkid:~$ sudo ovs-ofctl dump-flows -O OpenFlow13 s1
[sudo] password for captainkid:
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=300.543s, table=0, n_packets=0, n_bytes=0, idle_timeout=1800, priority=10000,
dl_type=0x88cc actions=output:32
 cookie=0x0, duration=146.182s, table=0, n_packets=4, n_bytes=472, idle_timeout=1800, priority=1001
,icmp6,in_port=3,ipv6_src=fe80::f82e:90ff:fe74:3fce,ipv6_dst=fe80::1,icmp_type=128 actions=output:2
 cookie=0x0, duration=146.178s, table=0, n_packets=4, n_bytes=472, idle_timeout=1800, priority=1002
,icmp6,in_port=2,ipv6_src=fe80::1,ipv6_dst=fe80::f82e:90ff:fe74:3fce,icmp_type=129 actions=output:3
 cookie=0x0, duration=300.543s, table=0, n_packets=28, n_bytes=2616, idle_timeout=1800, priority=0
actions=CONTROLLER:65535
captainkid@captainkid:~$ sudo ovs-ofctl dump-flows -O OpenFlow13 s2
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=350.347s, table=0, n_packets=0, n_bytes=0, idle_timeout=1800, priority=10000,
dl_type=0x88cc actions=output:32
 cookie=0x0, duration=195.992s, table=0, n_packets=4, n_bytes=472, idle_timeout=1800, priority=1001
,icmp6,in_port=3,ipv6_src=fe80::f82e:90ff:fe74:3fce,ipv6_dst=fe80::1,icmp_type=128 actions=output:2
 cookie=0x0, duration=195.981s, table=0, n_packets=4, n_bytes=472, idle_timeout=1800, priority=1002
,icmp6,in_port=2,ipv6_src=fe80::1,ipv6_dst=fe80::f82e:90ff:fe74:3fce,icmp_type=129 actions=output:3
 cookie=0x0, duration=350.347s, table=0, n_packets=30, n_bytes=2788, idle_timeout=1800, priority=0
actions=CONTROLLER:65535
```

Gambar 6.48 *Flow rules* pada *switch*

Gambar 6.48menunjukkan melalui perintah "ovs-ofctl dump-flows" pada *switch* "s2" dan "s1" bahwa *flow* dari koneksi *host* baru telah dapat terbentuk dimana *match* dari *flow* tersebut (IPv6 *destination*, IPv6 *source*, tipe ICMPv6 serta *action* yang dijalankan) disesuaikan oleh hasil *filtering* IPv6.

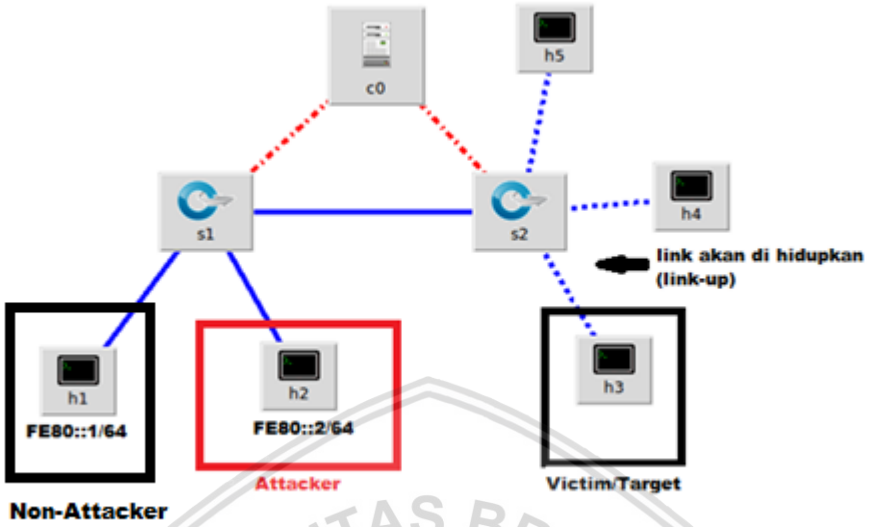
```
b6:62:f3:3d:90:56 (('b6:62:f3:3d:90:56', '-', '-', 'ICMP', 'BLOCK'),)
0e:3f:71:77:25:8f (('0e:3f:71:77:25:8f', '-', '-', 'ICMP', 'BLOCK'),)
ea:99:99:a9:0c:93 (('ea:99:99:a9:0c:93', '-', '-', 'ICMP', 'BLOCK'),)
```

Gambar 6.49 *ACL* rules pada kontroler

Gambar 6.49 menunjukkan bahwa kontroler dapat melakukan input entri ACL terhadap koneksi yang terdeteksi sebagai serangan dengan action 'BLOCK', dimana MAC *address* yang digunakan penyerang telah terdaftar di daftar *ACL Rules*.

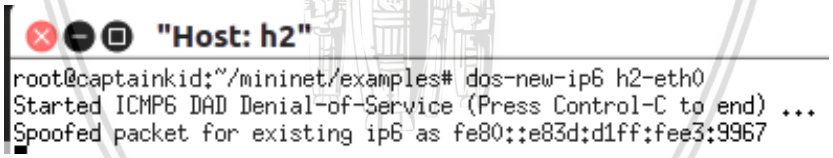
6.2.5 Pengujian *Duplicate Address Detection* DoS (Satu Non-Penyerang)

Pengujian ini dilakukan untuk menguji kinerja dari kemampuan *Stateful Packet Inspection* dalam membedakan *host* dari pelaku serangan *DAD DoS* dengan *host* yang bukan sebagai *host* penyerang. Pada pengujian ini, sistem akan diuji melalui skema *spoofing* yang dilakukan *host* H2 dimana penyerang berpura-pura menggunakan alamat yang ingin digunakan *host* baru yang mencoba menggunakan dua alamat IPv6 yang berbeda serta *host* H1 sebagai *host* yang bukan penyerang. Pengujian ini dijalankan guna melihat apakah sistem mampu melakukan identifikasi terhadap *host* yang ada.



Gambar 6.50 Topologi Pengujian

Dalam pelaksanaannya, pengujian dilakukan menggunakan topologi yang telah dibuat pada bab perancangan. *Link* yang statusnya aktif (*link-up*) adalah *link* pada *host* h1 dan h2, dimana *link* pada *host* lainnya (h3, h4 dan h5) diatur menjadi *down* (*link-down*) sehingga hanya *host* h1 dan h2 yang diawal telah aktif pada jaringan lokal. Simulasi serangan dilakukan dengan menjalankan tools THC-IPv6 toolkit pada *host* h2 dengan perintah "dos-new-ip6 h2-eth0".



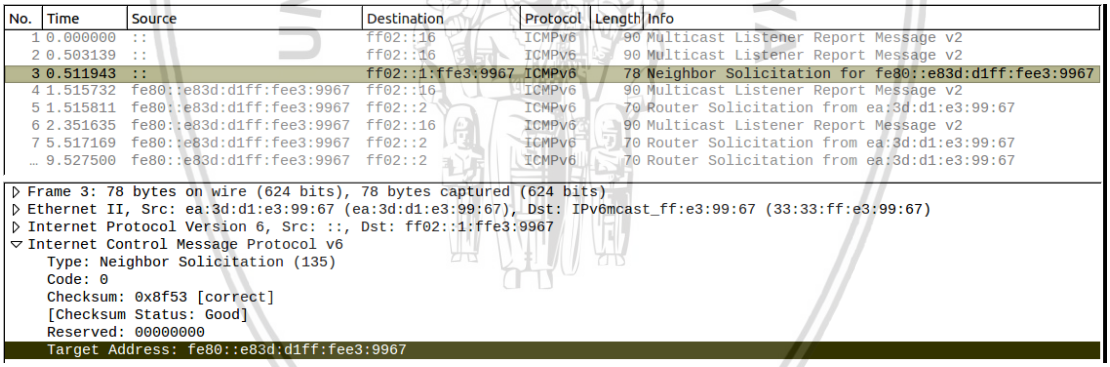
Gambar 6.51 *output tool* pada h2

Gambar 6.51 menunjukkan bahwa penyerang (h2) telah melakukan *spoofing* terhadap alamat IPv6 *host* h3 yang ingin terhubung ke dalam jaringan (*link-up*), yaitu "fe80::e83d:d1ff:fee3:9967".

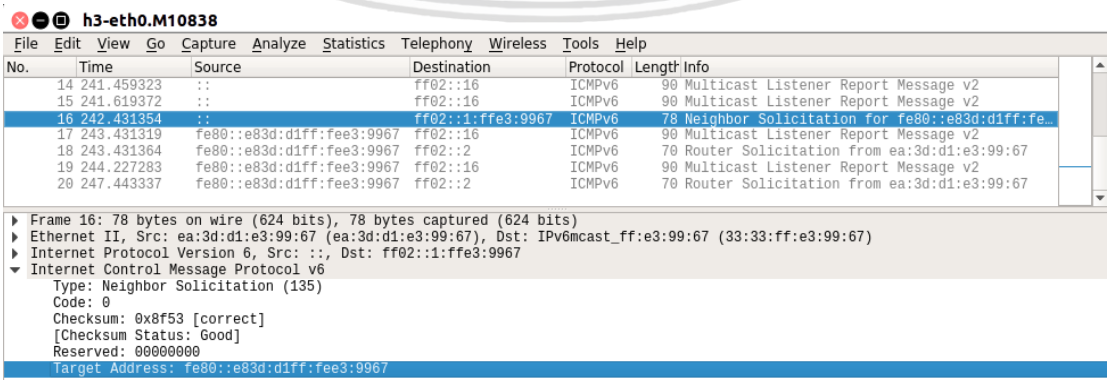
```
::  ->  ff02::1:ffe3:9967 : Neighbour Solicitation y
Input IP + MAC node baru ke state table
135 (('::', 'ea:3d:d1:e3:99:67'),)
::  ->  ff02::1:ffe3:9967 : Neighbour Solicitation y

135 (('::', 'ea:3d:d1:e3:99:67'),)
input alamat IPv6 baru pada node lama
135 (('fe80::e83d:d1ff:fee3:9967', 'ea:3d:d1:e3:99:67'),)
fe80::e83d:d1ff:fee3:9967  ->  ff02::1 : Neighbour Advertisement
Pengecekan Info dari Neighbor Advertisement
spoofed NA detection, drop NA packet 2

MAC Address  ->  6a:a4:4f:49:54:bb : entry ACL block list

fe80::e83d:d1ff:fee3:9967  ->  ff02::1 : Neighbour Advertisement
Pengecekan Info dari Neighbor Advertisement
spoofed NA detection, drop NA packet 2

MAC Address  ->  6a:a4:4f:49:54:bb : entry ACL block list
```

Gambar 6.52 *output controller* pada saat h3 terhubung

Pada gambar 6.52 terlihat bahwa terdapat 3 *state* koneksi yang tersimpan pada *state table* melalui hasil *filtering* oleh *Stateful Packet InspectionI* yang menandakan bahwa alamat IPv6 dari *host* 3 telah dikenali dan disimpan pada *state table*. Hal tersebut diperoleh dari inspeksi tiap koneksi yang terhubung. Berikut hasil capture Wireshark dari *host* baru (h3) dan *host* bukan penyerang (h1).

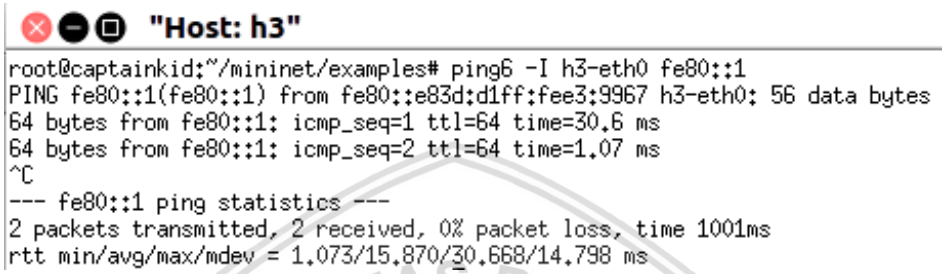


Gambar 6.53 capture Wireshark h1



Gambar 6.54 *capture* Wireshark h3

Gambar 6.53 memperlihatkan bahwa h1 menerima paket *neighbor* solicitation dari *host* baru yang menanyakan alamat "fe80::384d:d1ff:fee3:9967" dan tidak mengirim *neighbor advertisement* sebagai balasan terhadap *broadcast neighbor solicitation* yang dilakukan *host* baru. Gambar 6.54 menunjukkan hasil capture h3, dimana *host* h3 tidak menerima paket *neighbor advertisement* apapun. Hasil uji sistem ini dapat dipastikan dengan melakukan *ping* dari h3 ke h1 untuk menunjukkan apakah h3 sudah dapat terhubung dan berkomunikasi pada jaringan lokal.



Gambar 6.55 *ping* h3 ke h1

Gambar 6.55 menjelaskan bahwa *host* h3 dapat menggunakan alamat IPv6 yang ingin digunakan dengan menjalankan perintah ping dari *host* h3 menuju *host* h1). Sistem juga mampu menambahkan entri *ACL* (*Access Control List*) dari informasi koneksi yang terdeteksi sebagai suatu serangan yang disimpan didalam library *ACL*.

```
(('6a:a4:4f:49:54:bb', '-', '-', 'ICMP', 'BLOCK'),)
```

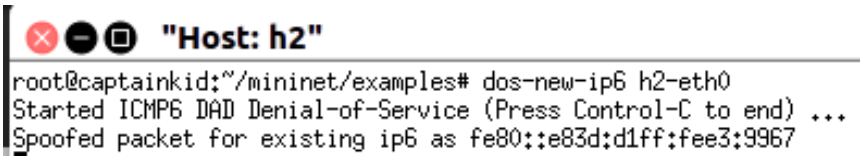
Gambar 6.56 *ACL Rules*

Gambar 6.56 menjelaskan bahwa kontroler dapat melakukan input entri ACL terhadap koneksi yang terdeteksi sebagai serangan dengan action 'BLOCK', dimana MAC *address* yang digunakan penyerang 6a:a4:4f:49:54:bb) telah terdaftar di daftar *ACL Rules*.

6.3 Analisis Hasil

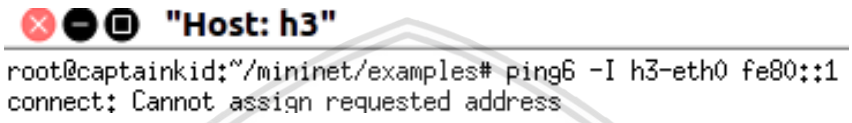
6.3.1 Analisis *Stateful Packet Inspection* Sebelum Pengembangan

Berdasarkan hasil pengujian *Stateful Packet Inspection* yang belum dilakukan pengembangan terhadap simulasi serangan *duplicate address detection DoS,* diperoleh hasil bahwa sistem belum mampu menangani serangan. Hal ini dapat dilihat dari hasil uji serangan, dimana *host* penyerang berhasil melakukan serangan pada *host* baru yang ingin menggunakan alamat IP baru.



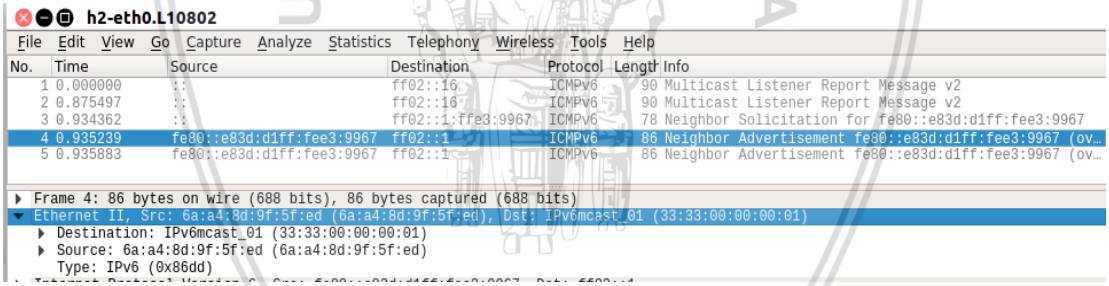
Gambar 6.57 *Host* penyerang

Pada gambar 6.57 terlihat bahwa *host* h2 melalui THC-IPv6 toolkit telah melakukan *spoofed packet* terhadap *neighbor* solicitation yang dikirim oleh h3 (*host* baru) dengan mengirim *spoofed neighbor advertisement* sebagai balasan *neighbor solicitation* dari *host* baru. Hal tersebut membuat *host* baru tidak dapat menggunakan alamat IP yang ingin dipakai.



Gambar 6.58 Pengujian ping

Pengujian ping menunjukkan bahwa h3 tidak dapat melakukan ping terhadap fe80::1 (*host* h1). Hal ini terjadi karena h3 tidak memiliki alamat yang dapat terhubung ke jaringan lokal. Pada gambar diatas ditunjukkan bahwa *host* baru (h3) tidak bisa menentapkan penggunaan alamat IP yang diminta h3. Hal ini terjadi karena sistem tidak memiliki *packet handle* NDP IPv6 yang mampu menangani serangan *DAD DoS.*



Gambar 6.59 Capture Wireshark h2

Simulasi serangan dapat dikatakan berhasil dikarenakan Pada gambar 6.59 terlihat bahwa h2 *host* penyerang mampu mengirimkan *spoofed neighbor advertisement* sebagai balasan dari *neighbor solicitation* yang dikirim oleh h3. Hal ini membuat *host* baru (h3) tidak dapat menggunakan alamat IPv6 yang ingin digunakan yang membuat *host* h3 tidak dapat terhubung pada jaringan lokal. Untuk itu diperlukan mekanisme untuk menangani proses *duplicate address detection* pada *Stateful Packet Inspection* yang telah ada.

6.3.2 Analisis *Stateful Packet Inspection* Setelah Pengembangan

Berdasarkan hasil pengujian *Stateful Packet Inspection* setelah dilakukan pengembangan terhadap simulasi serangan *duplicate address detection DoS* didapatkan bahwa sistem yang telah dikembangkan mampu menangani proses *duplicate address detection* serta serangan *duplicate address detection DoS.* Hal itu

terlihat dari hasil pengujian *duplicate address detection tanpa serangan,* serangan dengan satu penyerang maupun dengan dua penyerang.

6.3.2.1 *Analisis Pengujian Proses Duplicate Address Detection Tanpa Serangan*

Pada pengujian ini diperoleh hasil bahwa sistem mampu mengenali proses *duplicate address detection* dimana *host* baru yang membawa alamat IPv6 baru dapat terhubung kedalam jaringan lokal dimana *host* h1 dan h2 tidak membalas *broadcast neighbor solicitation* dari h3 tersebut dengan *neighbor advertisement* dikarenakan alamat yang ingin digunakan *host* h3 tersedia (tidak digunakan oleh *host* manapun di dalam jaringan).

```
::  ->  ff02::1:ffe3:9967 : Neighbour Solicitation y

135 (('fe80::1', '2e:60:86:c1:76:70'), ('fe80::2', '6a:a4:ca:04:da:b7')
)
Input IP + MAC node baru ke state table
135 (('fe80::1', '2e:60:86:c1:76:70'), ('fe80::2', '6a:a4:ca:04:da:b7')
, ('::', 'ea:3d:d1:e3:99:67'))
::  ->  ff02::1:ffe3:9967 : Neighbour Solicitation y

135 (('fe80::1', '2e:60:86:c1:76:70'), ('fe80::2', '6a:a4:ca:04:da:b7')
, ('::', 'ea:3d:d1:e3:99:67'))
input alamat IPv6 baru pada node lama
135 (('fe80::1', '2e:60:86:c1:76:70'), ('fe80::2', '6a:a4:ca:04:da:b7')
, ('fe80::e83d:d1ff:fee3:9967', 'ea:3d:d1:e3:99:67'))
```

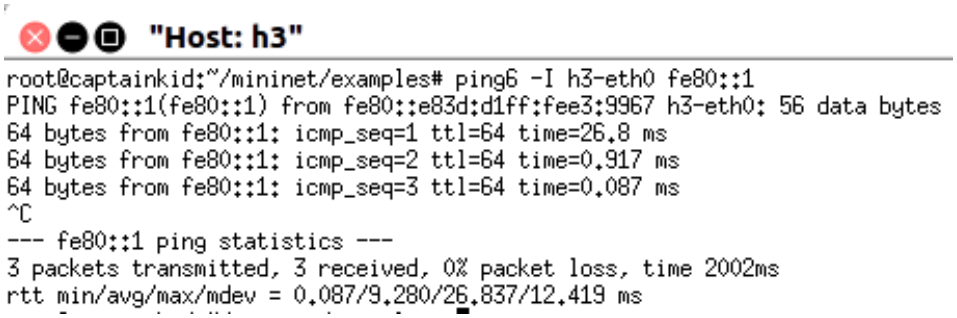
Gambar 6.60 Capture Kontroler

Pada gambar 6.60 terlihat bahwa kontroler mampu mengenali proses *duplicate address detection* dengan menangani paket *neighbor* solicitation dari *host* baru serta mampu membentuk dan memelihara *state table* berdasarkan alamat IPv6 serta alamat MAC dari tiap *host* yang terhubung. Dari output kontroler tersebut, diperoleh hasil dari *state table* sebagai berikut.

Tabel 6.1 *State table 1*

NDP	*IP source*	*MAC Source*
135	fe80::1	2e:60:86:c1:76:70
135	fe80::2	6a:a4:ca:04:da:b7
135	fe80::384d:d1ff:fee3:9967	Ea:3d:d1:e3:99:67

State table tersebut tersebut terbentuk berdasarkan pencocokan alamat IPv6 dan alamat MAC dari tiap *host* melalui mekanisme yang dijalankan oleh kontroler. *State table* tersebut akan berguna sebagai parameter inspeksi terhadap *host* baru yang ingin terhubung ke dalam jaringan lokal kedepannya.



Gambar 6.61 Pengujian ping6 antar *host*

Berdasarkan gambar 6.61 dapat terlihat pengujian ping6 dari *host* asal (h1) menuju ke *host* tujuan (h2) mampu mengirimkan 11 paket *ping* dengan rasio *packet* loss sebesar 0%. Pada saat paket *ping* pertama ditransmisikan, terdapat delay yang cendrung lebih besar dari paket seterusnya, hal itu dikarenakan pada saat paket pertama masuk, terjadi pembuatan dan penambahan entri *flow* ke dalam *flow table*. Dari hasil pengujian *ping* ini terlihat bahwa sistem dapat mengenali paket *ping* IPv6 (ICMPv6) serta mampu menjalankan *ping* antar *host* dengan menambahkan entri *flow* ke dalam *flow table* sesuai dengan informasi yang dibawa oleh paket tersebut.

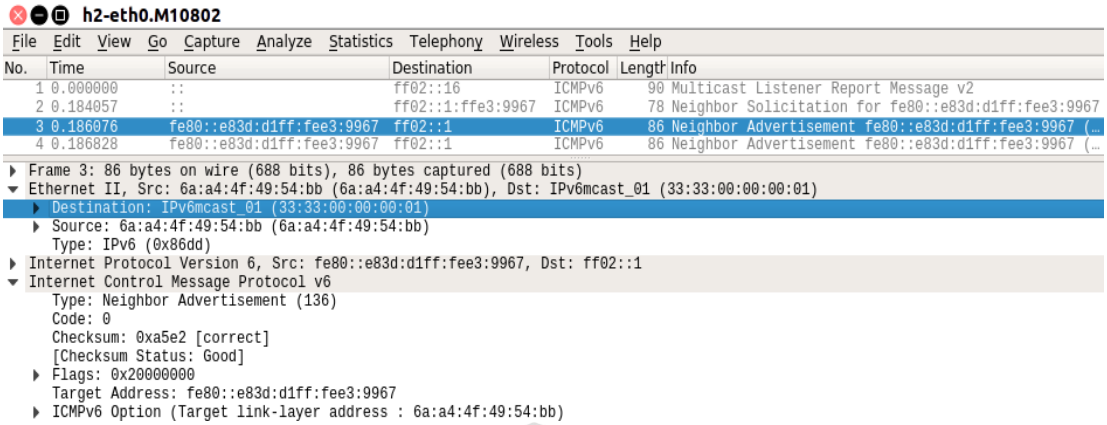
```
captainkid@captainkid:~$ sudo ovs-ofctl dump-flows -O OpenFlow13 s2
[sudo] password for captainkid:
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=515.308s, table=0, n_packets=2, n_bytes=236, idle_timeout=1800, priority=1001
,icmp6,in_port=1,ipv6_src=fe80::e83d:d1ff:fee3:9967,ipv6_dst=fe80::1,icmp_type=128 actions=output:2
 cookie=0x0, duration=515.296s, table=0, n_packets=2, n_bytes=236, idle_timeout=1800, priority=1002
,icmp6,in_port=2,ipv6_src=fe80::1,ipv6_dst=fe80::e83d:d1ff:fee3:9967,icmp_type=129 actions=output:1
 cookie=0x0, duration=2296.698s, table=0, n_packets=70, n_bytes=6658, idle_timeout=1800, priority=0
 actions=CONTROLLER:65535
```

Gambar 6.62 *Dump flow* dari *switch* s2

Berdasarkan gambar 6.62 dapat terlihat bahwa entri *flow* dari paket *ping* telah masuk berikut informasi detail yang dibawa paket tersebut (protokol, tipe ICMPv6 serta IP asal dan tujuan). Pembentukan *match* pada *flow rules* yang terbentuk diperoleh melalui hasil inspeksi dari informasi yang di bawa oleh tiap koneksi. Hasil ini menunjukkan bahwa sistem mampu membantu dalam pembentukan *flow rules* untuk efisiensi komunikasi pada jaringan lokal SDN.

6.3.2.2 *Analisis Pengujian Duplicate Address Detection DoS (Satu Penyerang)*

Pada pengujian ini diperoleh hasil bahwa sistem mampu mengenali proses *duplicate address detection* mencegah serangan *duplicate address detection DoS*, dimana *host* baru yang membawa alamat IPv6 baru dapat terhubung kedalam jaringan lokal.



Gambar 6.63 *Capture* Wireshark *h2*

Hasil pengujian pada gambar 6.63 menunjukkan *host* h2 selaku penyerang membalas *broadcast neighbor solicitation* dari h3 tersebut dengan melakukan *spoofing neighbor advertisement* dengan tujuan untuk menyamar sebagai *host* dengan alamat "fe80::e83d:d1ff:fee3:9967". Namun hal ini dapat dicegah oleh kontroler yang mampu mengenali serangan.

```
::  ->  ff02::1:ffe3:9967 : Neighbour Solicitation y
Input IP + MAC node baru ke state table
135 (('::', 'ea:3d:d1:e3:99:67'),)
::  ->  ff02::1:ffe3:9967 : Neighbour Solicitation y

135 (('::', 'ea:3d:d1:e3:99:67'),)
input alamat IPv6 baru pada node lama
135 (('fe80::e83d:d1ff:fee3:9967', 'ea:3d:d1:e3:99:67'),)
fe80::e83d:d1ff:fee3:9967  ->  ff02::1 : Neighbour Advertisement
Pengecekan Info dari Neighbor Advertisement
spoofed NA detection, drop NA packet 2
```

Gambar 6.64 *Output kontroler*

Gambar 6.64 menunjukkan sistem mendeteksi bahwa h2 melakukan *spoofing neighbor advertisement* sehingga paket *spoofing* tersebut tidak diteruskan oleh kontroler. Hal tersebut dapat dibandingkan dengan hasil pengujian serangan tanpa menggunakan *Stateful Packet Inspection* pada subbab sebelumnya, dimana pada subbab sebelumnya *neighbor advertisement* dari penyeragan masih dapat diterima oleh *host* baru. Sistem yang telah dikembangkan mampu menangani serangan *duplicate address detection DoS* dengan menangani paket *neighbor* solicitation dari *host* baru serta mampu membentuk dan memelihara *state table* berdasarkan alamat IPv6 serta alamat MAC dari tiap *host* yang terhubung. Dari output kontroler tersebut, diperoleh hasil dari *state table* sebagai berikut.

Tabel 6.2 *State table 2.1*

NDP	*IP source*	*MAC Source*
135	fe80::1	2e:60:86:c1:76:70
135	fe80::2	6a:a4:ca:04:da:b7
135	fe80::384d:d1ff:fee3:9967	ea:3d:d1:e3:99:67

State table tersebut tersebut terbentuk berdasarkan pencocokan alamat IPv6 dan alamat MAC dari tiap *host* melalui mekanisme yang dijalankan oleh kontroler. *State table* tersebut akan berguna sebagai parameter inspeksi terhadap *host* baru yang ingin terhubung ke dalam jaringan lokal kedepannya. Hasil pengujian selanjutnya terlihat bahwa sistem mampu memelihara *state table* serta mencegah serangan pada proses *duplicate address detection* yang dilakukakn oleh *host* h3, dimana *host* h3 mengubah alamat IPv6-nya menjadi "fe80::3"

```
::  ->  ff02::1:ff00:3 : Neighbour Solicitation y

135 (('fe80::1', '2e:60:86:c1:76:70'), ('fe80::3', 'ea:3d:d1:e3:99:67'))
IPv6 address valid/telah tersimpan
::  ->  ff02::1:ff00:3 : Neighbour Solicitation y

135 (('fe80::1', '2e:60:86:c1:76:70'), ('fe80::3', 'ea:3d:d1:e3:99:67'))
IPv6 address valid/telah tersimpan
fe80::3  ->  ff02::1 : Neighbour Advertisement
Pengecekan Info dari Neighbor Advertisement
spoofed NA detection, drop NA packet 2
```

Gambar 6.65 *Output kontroler*

Pada gambar 6.65 terlihat bahwa kontroler mampu menangani perubahan alamat IPv6 yang dilakukan oleh h3, yaitu dari alamat "fe80::384d:d1ff:fee3:9967" menjadi "fe80::3" dan mampu melakukan drop atas *neighbor advertisement* dari *host* h2 (penyerang). Dari hasil pemeliharaan *state table* yang dilakukan kontroler diperoleh nilai *state table* sabagai berikut.

Tabel 6.3 *State table 2.2*

NDP	*IP source*	*MAC Source*
135	fe80::1	2e:60:86:c1:76:70
135	fe80::2	6a:a4:ca:04:da:b7
135	fe80::3	ea:3d:d1:e3:99:67

Sistem mampu melakukan perubahan terhadap informasi yang ada pada *state table* melalui mekanisme yang telah diterapkan pada pengembangan *Stateful Packet Inspection*. Sistem juga mampu membentuk ACL list berdasarkan hasil inspeksi paket.

Tabel 6.4 *ACL Stateful Packer Inspection*

MAC	*Port Source*	*Port Destination*	*Packet Type*	*Action*
6a:a4:fc:7c:a9:5e	-	-	ICMP	BLOCK
6a:a4:4f:49:54:bb	-	-	ICMP	BLOCK

Pembentukan ACL diperoleh dari hasil inspeksi yang dilakukan sistem terhadap *host* yang diduga sebagai penyerang. Sehingga informasi dari paket yang dikirim oleh *host* penyerang dijadikan sebagai parameter pembentukan entri ACL. ACL ini nantinya digunakan sebagai parameter penyaringan pada saat paket masuk, sehingga *host* yang diduga penyerang yang telah terdaftar pada ACL tidak mampu terhubung pada jaringan lokal SDN.

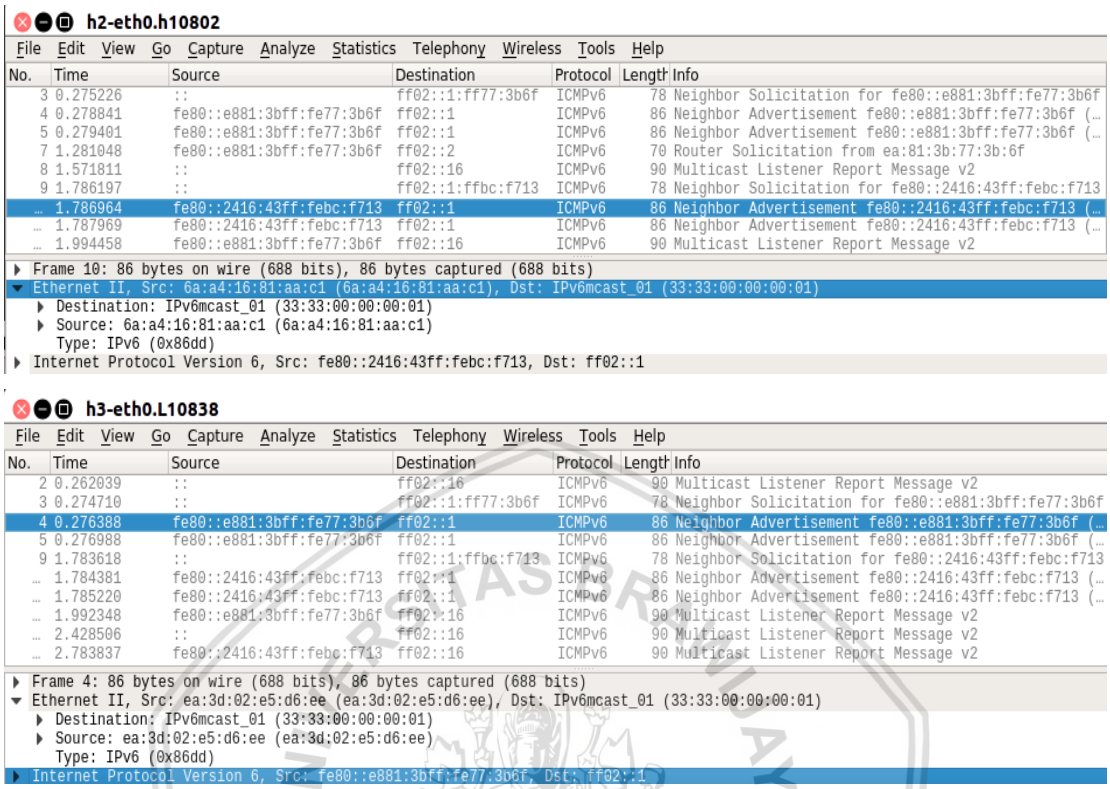
```
captainkid@captainkid:~$ sudo ovs-ofctl dump-flows -O OpenFlow13 s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=1692.827s, table=0, n_packets=0, n_bytes=0, idle_timeout=1800, priority=10000
,dl_type=0x88cc actions=output:32
 cookie=0x0, duration=149.831s, table=0, n_packets=2, n_bytes=236, idle_timeout=1800, priority=1001
,icmp6,in_port=3,ipv6_src=fe80::e881:3bff:fe77:3b6f,ipv6_dst=fe80::1,icmp_type=128 actions=output:2
 cookie=0x0, duration=149.827s, table=0, n_packets=2, n_bytes=236, idle_timeout=1800, priority=1002
,icmp6,in_port=2,ipv6_src=fe80::1,ipv6_dst=fe80::e881:3bff:fe77:3b6f,icmp_type=129 actions=output:3
 cookie=0x0, duration=111.497s, table=0, n_packets=2, n_bytes=236, idle_timeout=1800, priority=1001
,icmp6,in_port=3,ipv6_src=fe80::2416:43ff:febc:f713,ipv6_dst=fe80::1,icmp_type=128 actions=output:2
 cookie=0x0, duration=111.493s, table=0, n_packets=2, n_bytes=236, idle_timeout=1800, priority=1002
,icmp6,in_port=2,ipv6_src=fe80::1,ipv6_dst=fe80::2416:43ff:febc:f713,icmp_type=129 actions=output:3
 cookie=0x0, duration=1692.827s, table=0, n_packets=39, n_bytes=3576, idle_timeout=1800, priority=0
 actions=CONTROLLER:65535
```

Gambar 6.66 *Flow rules pada switch*

Berdasarkan gambar 6.66 dapat terlihat bahwa entri *flow* dari paket *ping* telah masuk berikut informasi detail yang dibawa paket tersebut (protokol, tipe ICMPv6 serta *IP* asal dan tujuan). Pembentukan *match* pada *flow rules* yang terbentuk diperoleh melalui hasil inspeksi dari informasi yang di bawa oleh tiap koneksi. Hasil ini menunjukkan bahwa sistem mampu membantu dalam pembentukan *flow rules* untuk efisiensi komunikasi pada jaringan lokal SDN.

6.3.2.3 *Analisis Pengujian Duplicate Address Detection DoS (Dua Penyerang)*

Pada pengujian ini diperoleh hasil bahwa sistem mampu mengenali proses *duplicate address detection* dan mencegah serangan *duplicate address detection DoS* melalui dua penyerang, dimana terdapat dua *host* baru yang membawa alamat IPv6 baru dapat terhubung kedalam jaringan lokal.



Gambar 6.67 *capture* Wireshark h2 dan h3

Hasil pengujian yang terlihat pada gambar 6.67 menunjukkan bahwa *host* h2 dan h3 selaku penyerang membalas *broadcast neighbor solicitation* dari h3 tersebut dengan melakukan *spoofing neighbor advertisement* dengan tujuan untuk menyamar sebagai *host* yang memiliki alamat IPv6 "fe80::2416:43ff:febc:f713" serta "fe80::3881:3bff:fe77:3b6f". Namun hal ini dapat dicegah oleh kontroler yang mampu mengenali serangan. Kontroler mampu melakukan *filtering* dengan memberikan notifikasi *drop spoofed packet* pada saat medeteksi adanya serangan yang dilakukan *host* h2 dan h3. Sistem juga mampu membentuk *state table* berdasarkan inspeksi dari hasil penyaringan paket sehingga terbentuk lima *state* koneksi yang tersimpan pada *state table* berdasarkan hasil inspeksi sistem, yaitu:

Tabel 6.5 *State table* kelima *host*

NDP	***IP source***	***MAC Source***
135	fe80::1	2e:60:86:c1:76:70
135	fe80::2	6a:a4:ca:04:da:b7
135	fe80::3	ea:3d:d1:e3:99:67
135	fe80::2416:43ff:febc:f713	26:16:43: Bc:f7:13
135	fe80::3881:3bff:fe77:3b6f	ea:81:3b:77:3b:6f

Proses *spoofing* yang dilakukan *host* penyerang (h2 dan h3) terhadap *host* baru dapat dicegah melalui proses filtering berdasarkan mekanisme pengecekan proses *duplicate address detection* serta inspeksi terhadap *state table* yang telah ada, sehingga sistem mampu mendeteksi apabila *attacker* mengirim paket *spoofing*. Sistem juga mampu membentuk ACL berdasarkan hasil inspeksi paket.

Tabel 6.6 *ACL Stateful Packer Inspection*

MAC	*Port Source*	*Port Destination*	*Packet Type*	*Action*
6a:a4:16:81:aa:c1	-	-	ICMP	BLOCK
6a:a4:04:76:ef:6d	-	-	ICMP	BLOCK
ea:3d:02:e5:d6:ee	-	-	ICMP	BLOCK
ea:3d:4d:d7:e1:2e	-	-	ICMP	BLOCK

Pembentukan ACL diperoleh dari hasil inspeksi yang dilakukan sistem terhadap *host* yang diduga sebagai penyerang. Sehingga informasi dari paket yang dikirim oleh *host* penyerang dijadikan sebagai parameter pembentukan entri ACL. ACL ini nantinya digunakan sebagai parameter penyaringan pada saat paket masuk, sehingga *host* yang diduga penyerang yang telah terdaftar pada ACL tidak mampu terhubung pada jaringan lokal SDN.

```
captainkid@captainkid:~$ sudo ovs-ofctl dump-flows -O OpenFlow13 s2
[sudo] password for captainkid:
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=1654.570s, table=0, n_packets=0, n_bytes=0, idle_timeout=1800, priority=10000
,dl_type=0x88cc actions=output:32
 cookie=0x0, duration=111.582s, table=0, n_packets=2, n_bytes=236, idle_timeout=1800, priority=1001
,icmp6,in_port=4,ipv6_src=fe80::e881:3bff:fe77:3b6f,ipv6_dst=fe80::1,icmp_type=128 actions=output:2
 cookie=0x0, duration=111.569s, table=0, n_packets=2, n_bytes=236, idle_timeout=1800, priority=1002
,icmp6,in_port=2,ipv6_src=fe80::1,ipv6_dst=fe80::e881:3bff:fe77:3b6f,icmp_type=129 actions=output:4
 cookie=0x0, duration=73.248s, table=0, n_packets=2, n_bytes=236, idle_timeout=1800, priority=1001,
icmp6,in_port=3,ipv6_src=fe80::2416:43ff:febc:f713,ipv6_dst=fe80::1,icmp_type=128 actions=output:2
 cookie=0x0, duration=73.235s, table=0, n_packets=2, n_bytes=236, idle_timeout=1800, priority=1002,
icmp6,in_port=2,ipv6_src=fe80::1,ipv6_dst=fe80::2416:43ff:febc:f713,icmp_type=129 actions=output:3
 cookie=0x0, duration=1654.570s, table=0, n_packets=39, n_bytes=3576, idle_timeout=1800, priority=0
 actions=CONTROLLER:65535_

captainkid@captainkid:~$ sudo ovs-ofctl dump-flows -O OpenFlow13 s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=1692.827s, table=0, n_packets=0, n_bytes=0, idle_timeout=1800, priority=10000
,dl_type=0x88cc actions=output:32
 cookie=0x0, duration=149.831s, table=0, n_packets=2, n_bytes=236, idle_timeout=1800, priority=1001
,icmp6,in_port=3,ipv6_src=fe80::e881:3bff:fe77:3b6f,ipv6_dst=fe80::1,icmp_type=128 actions=output:2
 cookie=0x0, duration=149.827s, table=0, n_packets=2, n_bytes=236, idle_timeout=1800, priority=1002
,icmp6,in_port=2,ipv6_src=fe80::1,ipv6_dst=fe80::e881:3bff:fe77:3b6f,icmp_type=129 actions=output:3
 cookie=0x0, duration=111.497s, table=0, n_packets=2, n_bytes=236, idle_timeout=1800, priority=1001
,icmp6,in_port=3,ipv6_src=fe80::2416:43ff:febc:f713,ipv6_dst=fe80::1,icmp_type=128 actions=output:2
 cookie=0x0, duration=111.493s, table=0, n_packets=2, n_bytes=236, idle_timeout=1800, priority=1002
,icmp6,in_port=2,ipv6_src=fe80::1,ipv6_dst=fe80::2416:43ff:febc:f713,icmp_type=129 actions=output:3
 cookie=0x0, duration=1692.827s, table=0, n_packets=39, n_bytes=3576, idle_timeout=1800, priority=0
 actions=CONTROLLER:65535
```

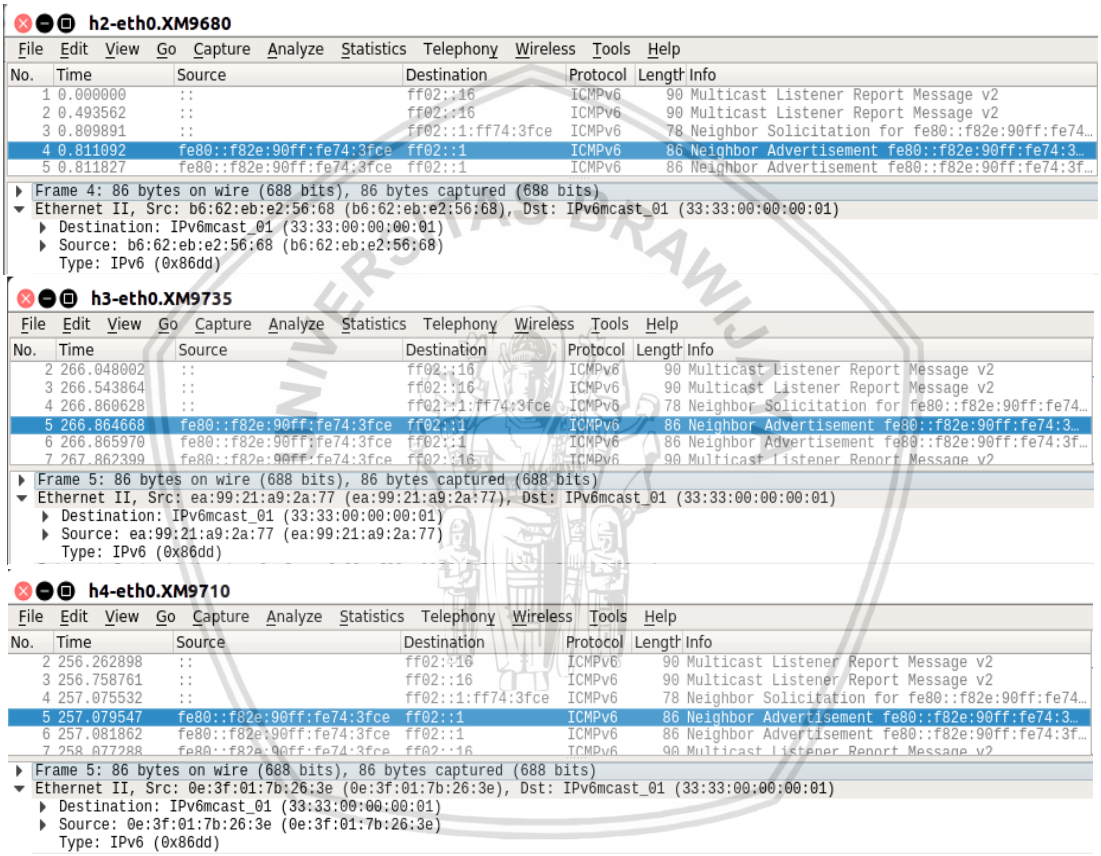
Gambar 6.68 *Flow rules* pada *switch*

Berdasarkan gambar 6.68 dapat terlihat bahwa entri *flow* dari paket *ping* telah masuk berikut informasi detail yang dibawa paket tersebut (protokol, tipe ICMPv6 serta *IP* asal dan tujuan). Pembentukan *match* pada *flow rules* yang terbentuk

diperoleh melalui hasil inspeksi dari informasi yang di bawa oleh tiap koneksi. Hasil ini menunjukkan bahwa sistem mampu membantu dalam pembentukan *flow rules* untuk efisiensi komunikasi pada jaringan lokal SDN.

6.3.2.4 *Analisis Pengujian Duplicate Address Detection DoS (TIga Penyerang)*

Pada pengujian ini diperoleh hasil bahwa sistem mampu mengenali proses *duplicate address detection* dan mencegah serangan *duplicate address detection DoS* melalui dua penyerang, dimana terdapat dua *host* baru yang membawa alamat IPv6 baru dapat terhubung kedalam jaringan lokal.



Gambar 6.69 *capture* Wireshark h2, h3 dan h4

Hasil pengujian yang terlihat pada gambar 6.69 menunjukkan bahwa *host* h2, h3 dan h4 selaku penyerang membalas *broadcast neighbor solicitation* dari h3 tersebut dengan melakukan *spoofing neighbor advertisement* dengan tujuan untuk menyamar sebagai *host* yang memiliki alamat IPv6 “fe80::f82e:90ff:fe74:3fce”. Namun hal ini dapat dicegah oleh kontroler yang mampu mengenali serangan. Kontroler mampu melakukan *filtering* dengan memberikan notifikasi *drop spoofed packet* pada saat mendeteksi adanya serangan yang dilakukan *host* h2, h3 dan h4. Sistem juga mampu membentuk *state table* berdasarkan inspeksi dari hasil

penyaringan paket sehingga terbentuk lima *state* koneksi yang tersimpan pada *state table* berdasarkan hasil inspeksi sistem, yaitu:

Tabel 6.7 *State table* kelima *host*

NDP	*IP source*	*MAC Source*
135	fe80::1	2e:60:86:c1:76:70
135	fe80::2	6a:a4:ca:04:da:b7
135	fe80::3	ea:3d:d1:e3:99:67
135	fe80::2416:43ff:febc:f713	26:16:43: Bc:f7:13
135	fe80::f82e:90ff:fe74:3fce	56:bc:ba:b9:7a:9b

Proses *spoofing* yang dilakukan *host* penyerang (h2, h3 dan h4) terhadap *host* baru dapat dicegah melalui proses filtering berdasarkan mekanisme pengecekan proses *duplicate address detection* serta inspeksi terhadap *state table* yang telah ada, sehingga sistem mampu mendeteksi apabila *attacker* mengirim paket *spoofing*. Sistem juga mampu membentuk ACL berdasarkan hasil inspeksi paket.

Tabel 6.8 *ACL Stateful Packer Inspection*

MAC	*Port Source*	*Port Destination*	*Packet Type*	*Action*
b6:62:f3:3d:90:56	-	-	ICMP	BLOCK
0e:3f:71:77:25:8f	-	-	ICMP	BLOCK
ea:99:99:a9:0c:93	-	-	ICMP	BLOCK

Pembentukan ACL diperoleh dari hasil inspeksi yang dilakukan sistem terhadap *host* yang diduga sebagai penyerang. Sehingga informasi dari paket yang dikirim oleh *host* penyerang dijadikan sebagai parameter pembentukan entri ACL. ACL ini nantinya digunakan sebagai parameter penyaringan pada saat paket masuk, sehingga *host* yang diduga penyerang yang telah terdaftar pada ACL tidak mampu terhubung pada jaringan lokal SDN.

```
captainkid@captainkid:~$ sudo ovs-ofctl dump-flows -O OpenFlow13 s1
[sudo] password for captainkid:
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=300.543s, table=0, n_packets=0, n_bytes=0, idle_timeout=1800, priority=10000,
dl_type=0x88cc actions=output:32
 cookie=0x0, duration=146.182s, table=0, n_packets=4, n_bytes=472, idle_timeout=1800, priority=1001
,icmp6,in_port=3,ipv6_src=fe80::f82e:90ff:fe74:3fce,ipv6_dst=fe80::1,icmp_type=128 actions=output:2
 cookie=0x0, duration=146.178s, table=0, n_packets=4, n_bytes=472, idle_timeout=1800, priority=1002
,icmp6,in_port=2,ipv6_src=fe80::1,ipv6_dst=fe80::f82e:90ff:fe74:3fce,icmp_type=129 actions=output:3
 cookie=0x0, duration=300.543s, table=0, n_packets=28, n_bytes=2616, idle_timeout=1800, priority=0
actions=CONTROLLER:65535
captainkid@captainkid:~$ sudo ovs-ofctl dump-flows -O OpenFlow13 s2
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=350.347s, table=0, n_packets=0, n_bytes=0, idle_timeout=1800, priority=10000,
dl_type=0x88cc actions=output:32
 cookie=0x0, duration=195.992s, table=0, n_packets=4, n_bytes=472, idle_timeout=1800, priority=1001
,icmp6,in_port=3,ipv6_src=fe80::f82e:90ff:fe74:3fce,ipv6_dst=fe80::1,icmp_type=128 actions=output:2
 cookie=0x0, duration=195.981s, table=0, n_packets=4, n_bytes=472, idle_timeout=1800, priority=1002
,icmp6,in_port=2,ipv6_src=fe80::1,ipv6_dst=fe80::f82e:90ff:fe74:3fce,icmp_type=129 actions=output:3
 cookie=0x0, duration=350.347s, table=0, n_packets=30, n_bytes=2788, idle_timeout=1800, priority=0
actions=CONTROLLER:65535
```

Gambar 6.70 *Flow rules* pada *switch*

Berdasarkan gambar 6.70 dapat terlihat bahwa entri *flow* dari paket *ping* telah masuk berikut informasi detail yang dibawa paket tersebut (protokol, tipe ICMPv6 serta *IP* asal dan tujuan). Pembentukan *match* pada *flow rules* yang terbentuk diperoleh melalui hasil inspeksi dari informasi yang di bawa oleh tiap koneksi. Hasil ini menunjukkan bahwa sistem mampu membantu dalam pembentukan *flow rules* untuk efisiensi komunikasi pada jaringan lokal SDN.

6.3.2.5 *Pengujian Duplicate Address Detection DoS (Satu Non-Penyerang)*

Pada pengujian ini diperoleh hasil bahwa sistem mampu mengenali *host* yang bukan sebagai penyerang pada proses *duplicate address detection*. *Host* baru yang membawa alamat IPv6 baru dapat terhubung kedalam jaringan lokal dimana *host* h1 tidak membalas *broadcast neighbor solicitation* dari h3 tersebut dengan *neighbor advertisement* dikarenakan alamat yang ingin digunakan *host* h3 tersedia (tidak digunakan oleh *host* manapun di dalam jaringan).

```
::  ->  ff02::1:ffe3:9967 : Neighbour Solicitation y

135 (('fe80::1', '2e:60:86:c1:76:70'), ('fe80::2', '6a:a4:ca:04:da:b7')
)
Input IP + MAC node baru ke state table
135 (('fe80::1', '2e:60:86:c1:76:70'), ('fe80::2', '6a:a4:ca:04:da:b7')
, ('::', 'ea:3d:d1:e3:99:67'))
::  ->  ff02::1:ffe3:9967 : Neighbour Solicitation y

135 (('fe80::1', '2e:60:86:c1:76:70'), ('fe80::2', '6a:a4:ca:04:da:b7')
, ('::', 'ea:3d:d1:e3:99:67'))
input alamat IPv6 baru pada node lama
135 (('fe80::1', '2e:60:86:c1:76:70'), ('fe80::2', '6a:a4:ca:04:da:b7')
, ('fe80::e83d:d1ff:fee3:9967', 'ea:3d:d1:e3:99:67'))
```

Gambar 6.71 Capture Wireshark h2

Pada gambar 6.71 terlihat bahwa kontroler mampu mengenali proses *duplicate address detection* dengan menangani paket *neighbor* solicitation dari *host* baru serta

mampu membentuk dan memelihara *state table* berdasarkan alamat IPv6 serta alamat MAC dari tiap *host* yang terhubung. Dari output kontroler tersebut, diperoleh hasil dari *state table* sebagai berikut.

6.3.2.6 *Analisis* Rasio Keberhasil Sistem Terhadap *Duplicate Address Detection DoS*

Melalui hasil analisis sistem terhadap serangan *duplicate address detection* DoS, maka diperoleh rasio keberhasilan sistem dengan melakukan kalkulasi dari keseluruhan skema pengujian serangan. Pada pengujian serangan pertama, yaitu pengujian *duplicate address detection* DoS (satu penyerang) yang dimana terdapat satu *host* sebagai penyerang dan satu *host* sebagai *host* baru, didapatkan hasil bahwa sistem berhasil mencegah serangan. Pada pengujian serangan kedua, yaitu pengujian *duplicate address detection* DoS (dua penyerang) yang dimana terdapat dua *host* sebagai penyerang dan dua *host* sebagai *host* baru, didapatkan hasil bahwa sistem berhasil mencegah serangan. Pada pengujian serangan ketiga, yaitu pengujian *duplicate address detection* DoS (tiga Penyerang) yang dimana terdapat tiga *host* sebagai penyerang dan satu *host* sebagai *host* baru, melalui 10 percobaan pada tiap skenario didapatkan hasil bahwa sistem berhasil mencegah serangan.

Tabel 6.9 Keberhasilan Sistem

Urutan	Tipe Serangan	Keterangan
Serangan pertama	Menggunakan 1 *host* penyerang & 1 *host* baru (*link*-up)	Sistem Berhasil mencegah
	Menggunakan 1 *host* penyerang & 1 *host* lama (set IPv6 baru)	Sistem Berhasil mencegah
Serangan kedua	Menggunakan 2 *host* penyerang & 2 *host* baru (*link*-up)	Sistem Berhasil mencegah
Serangan ketiga	Menggunakan 3 *host* penyerang & 1 *host* baru (*link*-up)	Sistem Berhasil mencegah

Berdasarkan tabel diatas, diperoleh hasil bahwa sistem mampu menangani paket serangan dari seluruh skenario pengujian yang dilakukan dengan perulangan sebanyak 10 kali pada tiap skenario dalam mengatasi serangan *duplicate address detection* DoS serta mengenali *host* yang merupakan penyerang maupun yang bukan penyerang.

BAB 7
PENUTUP

7.1 Kesimpulan

Berdasarkan penelitian yang dilakukan terhadap pengembangan sistem yang telah di implementasi, maka dapat ditarik kesimpulan dari penelitian ini sebagai berikut.

1. Penulis telah melakukan pengembangan *Stateful Packet Inspection* dengan mengadaptasi mekanisme NDPmon guna mengatasi permasalahan dalam keamanan pada jaringan OpenFlow, khususnya terhadap celah pada proses *duplicate address detection*. Dalam penerapannya, sistem melakukan penyaringan dari tipe paket ICMPv6 yang masuk, dimana tipe paket ICMPv6 *neighbor solicitation* akan diteruskan ke *neighbor solicitation* untuk memperoleh informasi dalam pembentukan *state table.* Tipe paket ICMPv6 neighbor advertisement diteruskan ke *neighbor advertisement*, dimana sistem melakukan pengecekan IP-MAC dari paket tersebut terhadap *state table* yang telah ada sebelum diteruskan ke alamat tujuan.
2. Setelah dilakukan pengujian dan analisis, sistem mampu mengenali proses IPv6 *duplicate address detection* yang dilakukan oleh *host* baru pada jaringan OpenFlow, dimana sistem mampu membentuk *state table* yang berisi informasi dari penggunaan alamat IPv6 serta alamat MAC perangkat dari tiap *host* yang terhubung. Sistem juga mampu mengenali proses serangan IPv6 *DAD DoS,* dimana saat terjadi serangan sistem mampu melakukan *action* yang tepat terhadap paket yang dikirim oleh penyerang. Melalui tiga scenario pengujian serangan, diperoleh hasil sistem berhasil menangani setiap paket serangan dari seluruh skenario pengujian dengan mealkuakn perulangan 10 kali dalam mengatasi serangan *DAD DoS* serta mampu mengenali *host* yang merupakan penyerang maupun yang bukan penyerang.

7.2 Saran

Beberapa saran berdasarkan hasil penelitian ini untuk pengembangan penelitian selanjutnya adalah sebagai berikut.

1. Perlu dilakukan analisis performa dari pengembangan *Stateful Packet Inspection* berdasarkan jumlah *node* (*host dan switch)* yang masif untuk memperoleh batas efisiensi sistem.
2. Perlu dilakukan penelitian terkait dengan menerapkan sistem pada jaringan fisik asli (*switch* Openflow).

DAFTAR PUSTAKA

InMon. (2017). *sFlow-RT*. Diambil kembali dari inMon: The Inventors of s*Flow*: http://www.inmon.com/products/s*Flow*-RT.php

Kreutz, D., Verissimo, P. E., Azodolmolky, S., Ramos, F. M., Rothenberg, C. E., & Uhlig, S. (2015). Software-Defined *Network*ing: A Comprehensive Survey. *Proceedings of the IEEE.* IEEE

Zhang, Zhibin. (2015). *LASF: A Flow Scheduling Policy in Stateful Packet Inspection Systems.* IEEE

Othman, Wajdy. (2017). *Implementation and Performance Analysis of* SDN *Firewall on POX Controller.* IEEE

Jayawardhana , Madusha. (2016). *Implementation of Stateful* SDN *Firewall*. IEEE

Astuto, B. N., Mendon¸ca, M., Nguyen, X. N., Obraczka, K., & Turletti, T. (2014). A Survey of Software-Defined *Network*ing: Past, Present, and Future of Programmable *Network*s. *Communications Surveys and Tutorials*. IEEE Communications Society, Institute of Electrical and Electronics Engineers.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., . . . Turner, J. (2008). OpenFlow*: Enabling Innovation in Campus Networks.* OpenFlow.

Open *Network*ing Fouᵲcation. (2013). OpenFlow *Switch* Specification *Version* 1.3.2. Diambil kembali dari OpenFlow: https://www.open*network*ing.org/images/stories/downloads/SDN-re*source*s/onf-specifications/OpenFlow/OpenFlow-spec-v1.3.2.pdf

Rouse, M. (2012, November). SDN *controller (software-defined networking controller)*. Diambil kembali dari TechTarget: SearchSDN: http://searchSDN.techtarget.com/definition/SDN-*controller*-software-defined-*network*ing-*controller*

Mininet Team. (2016). Mininet *Overview*. Diambil kembali dari Mininet: http://Mininet.org/overview/

Lantz, B., Heller, B., & McKeown, N. (2010). A *Network* in a Laptop: Rapid Prototy*ping* for Software-Defined *Network*s. *SIGCOMM.* New Delhi: ACM.

Ryu SDN *Framework* Community. (2014). Ryu*: Build* SDN *Agilely*. (Ryu) Dipetik January 24, 2017, dari https://osrg.github.io/Ryu/

Gont, F. (2012). Security *Assessment* of IPv6 *Neighbor* Discovery Implementations.

Cisco Team. (2007). The *Internet Protocol* Journal, Cisco Systems. San Jose: USA.

Barbhuiya, F., Biswas, S., Nandi, S. (2013). *Detection of Neighbor solicitation and AdvertisementSpoofing in IPv6 Neighbor Discovery Protocol*. IEEE

Biondi, Philippe. (2018). *Scapy Documentation, Release 2.4.0-dev*.

Diambil kembali dari Readthedocs *Scapy*: https://media.readthedocs.org/pdf/*scapy*/latest/*scapy*.pdf

Sahi, Aqeel. (2017). *An Efficient DDoS TCP Flood Attack Detection and Prevention System in a Cloud Environment*. IEEE

Hegr, Tomas. (2013). *OpenFlow Deployment and Concept Analysis.* AEEE

Deering, S. (2017). *Internet Protocol, Version 6 (*IPv6*) Specification*. IETF

Tseng, Chia-Wei. (2014). *IPv6 Operations and Deployment Scenarios over* SDN. IEICE

Sun, Weifeng. (2016). *A Collaborated IPv6-Packets Matching Mechanism Base on Flow Label in OpenFlow*. LNICST